

Advantage[™] VISION:Results[™] for z/OS

Reference Guide

r6



This documentation and any related computer software help programs (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of the Documentation for their own internal use, and may make one copy of the related software as reasonably required for back-up and disaster recovery purposes, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the provisions of the license for the product are permitted to have access to such copies.

The right to print copies of the Documentation and to make a copy of the related software is limited to the period during which the applicable license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

EXCEPT AS OTHERWISE STATED IN THE APPLICABLE LICENSE AGREEMENT, TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.

The use of any product referenced in the Documentation is governed by the end user's applicable license agreement.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Copyright © 2007 CA. All rights reserved.

Contents

Chapter 1: Introduction

Contacting CA Technical Support.	30

Chapter 2: Field and Work Area Definitions

Field Definitions	31
Data Name	32
Field Length	33
Starting Location	33
FILLER Keyword.	34
Redefining a Field.	34
Data Type	34
Number of Decimals	35
VALUE nnn `xxx' X`nn' ALL NULL LOWVALUES HIGHVALUES.	35
Numeric Values nnn	35
Character Values `xxx'	35
Hexadecimal Values X`nn'	36
ALL Values	36
NULL Values	36
LOWVALUES Values	37
HIGHVALUES Values.	37
REINIT	37
Extended Format	38
Edit Code	38
Report Output Field Size	38
ROUND or ROUNDED	39
Column Heading.	40
LEFT or RIGHT	40
SUM or NOSUM	40
Work Areas	41
REDEFINE	43
Special Fields and Reserved Words	44
BLANK or BLANKS	45
DYLCENTRY1	45
DYLCENTRY2	46
DYLCNAME1 Through DYLCNAME99, DYLCNAME00, DYLCNAME0A Through DYLCNAME0Z.	46

Character/Binary	60
Numeric/Numeric.	60
Numeric/Binary	60
Numeric/Packed.	60
Packed/Packed	60
Packed/Binary	60
Binary/Binary	61
Rules for Moving Fields	61
Character/Character.	61
Character/Numeric.	61
Character/Packed.	61
Character/Binary	61
Numeric/Character.	61
Numeric/Numeric.	62
Numeric/Packed.	62
Numeric/Binary	62
Packed/Character.	62
Packed/Numeric.	62
Packed/Packed	62
Packed/Binary	63
Binary/Character	63
Binary/Numeric	63
Binary/Packed	63
Binary/Binary	63
Literals	63
Character or Alphanumeric Literals.	64
Moving Character Literals	65
Comparing Character Literals	65
Assigning Character Values to Fields	65
Listing Literals in a Report.	66
Printing a Literal in a FILE PRINT	66
Numeric Literals.	67
Moving Numeric Literals	67
Comparing Numeric Literals	67
Using Numeric Literals in Arithmetic Expressions	68
Assigning Values to Numeric, Packed, or Binary Fields	68
Hexadecimal Literals	68
Moving Hexadecimal Literals	68
Comparing Hexadecimal Literals	69
Assigning Values to Fields Using Hexadecimal Notation.	69

Chapter 4: Using the OPTION Statement

Program Input and Processing Options	73
CDLOAD Facility	73
Input Columns	73
Programming Mode	74
Input Literal Data	74
Old Release Keyword Equivalents	75
Modifying IF NUMERIC Test	75
IF NUMERIC Test for PD Fields	75
Qualification of Data Names	75
Allocating Memory for Tables and Arrays	76
VSE Disk Work File	76
User-Defined Index Fields	77
VSAM File Characteristics	77
Match File Attributes for VSE Sequential Files	78
Divide by Zero Condition	78
Program Output Options	79
COBAPOS Option	79
COBXSIGN Option	79
Currency Symbol Substitution	79
Four-Digit Year Support	79
FILE PRINT Options	80
Null Output File Control	81
PDS Replace	81
Counter Display	81
Extended Arithmetic Report Field Display	81
VSE Report File Name	82
System-Determined Block Size Support for z/OS	82
DYLTIME Format Change	82
Environment Specification Options	83
COBOL II Environment	83
IBM Language Environment (LE) Support	83
AllFusion® Endeavor® Environment Variables	83
Options for Advantage™ VISION:Results® Interface™ to DB2	84
Use DB2 Table Definitions	84
For VISION:Results Interface to DB2 (Dynamic)	84
For VISION:Results Interface to DB2 (Static)	84
Resource Optimization Options	84
Optimization	85
Companion Product Options	86
VISION:Sixty	86
VISION:Excel Keywords	87
VISION:Excel Display	87
VSE Spooling VISION:Excel Output	87
VISION:Inquiry Bridge	88
VISION:Report	88

Program Listing and Formatting Options	88
Number of Lines per Page	89
Default Options Page Display	89
Automatic Control Totals	89
Debugging	89
Cross-Reference Listings	90
ABEND Analysis	91
Expand Error Message Information	91
Premature Sort Termination Without Abend	91
Generated Object Code Listing	91
Print Entry Point Address	91
Extended Error Analysis	92
Verify Syntax	92
Options Useful for Technical Support	92

Chapter 5: Data Name Qualification

Qualification Format	94
Qualifying of Keywords and Self-Defining Data Names	94
Qualifying Work Areas	95
Qualifying with Multiple Reports	95
MOVE CORRESPONDING and OPTION QLF	96
Options XREF/XREFA and DMAP with QLF	97
XREFA Option	97
DMAP Option	98
No Qualified Data Names Option	99

Chapter 6: Using the FILE Command

Supported Access Methods	103
BDAM (or DAM) Files	104
EXIT Files	105
ISAM Files	105
PDS Files	106
SEQUENTIAL Files	107
SSL Files	107
VSAM ESDS Files	108
VSAM KSDS Files	108
VSAM RRDS Files	109
Syntax Definitions (General)	110
Filename	110
File Organization	111
Input/Output Type	111
Processing Mode	111

SEQUENTIAL	111
FROM filename	112
Record Format	112
Record Size	113
Block Size	114
STATUS dataname	114
LENGTH dataname	115
COUNT dataname	116
CARD or CARDS	117
JCL	118
INTERPRET	118
SELECT dataname	118
INTSEL dataname	119
BYPASS nn	119
DUMMY	119
NULL	120
DROPERR	120
SYSnnn	121
Device Specifications	121
TAPE Label Rewind (Options)	121
Multiple File Input	122
TAPE nn	122
TAPE NUMFILES or DISKETTE NUMFILES	122
DISK nnnn	123
DISKETTE (options)	124
ONEBUFF	124
DYNAM	124
ISAM and VSAM Syntax Definitions	125
KEYLEN nnn	125
KEYLOC nnnn	125
STATUS dataname	125
POSITION dataname	126
PARTKEY nnn	126
REUSE	127
ERASE dataname	127
RANDOM dataname (data type) and RANDOMX dataname	127
SKIP dataname (data type) and SKIPX dataname	128
RELBYTE dataname and RELBYTEX dataname	129
PASSWORD 'password'	130
PASSWORD dataname	130
CYLOFL nn	130
EXTENTS nn	131
MASTER	131
RETAIN	131
EXIT and MODIFY	131
MODIFY modulename [nK]	131
Restrictions	132
EXIT modulename [nK]	132

Restrictions	133
SIZE nnnnn	134
MODIFY	134
EXIT	134
PARM (dataname)	135
IOAREA	135
WORK	135
IOU-Processed Files	136
Considerations for Processing Variable-Length Records	136
Record Size	136
Block Size	137
Location of Data Within Record	137
Length of Current Record	137
Output from Itself	137
Output from an Input File	138
File Printing Variable Files	138
Sorting a Variable File	138
Exceptions to the Above Rules	139
z/OS	139
Considerations for Unblocked ISAM Files	140
Sequential Read	140
Sequential Load/Extend	140
Random Retrieval	140
Random Update	141
Considerations for Concatenated Input Files	141

Chapter 7: Using the IF Command

Comparing and Selecting Data	143
IF	144
NOT	144
Relational Operators	144
Dataname or Literal	145
True Condition-1 Imperatives	145
False Condition Imperatives	146
ENDIF	146
Range Compares	146
Series Compares	147
Combined Range and Series Compares	147
Class or Status Tests	147
Test for Positive Value	147
Test for Negative Value	148
Test for Numeric Value	148
Test for Bits On	149
Test for Bits Mixed	150
Compound Compares	150
Select or Process if Any of Several Conditions Is True	151
Select or Process if All of Several Conditions Are True	152
Combining Compound Compares	152

Interval and Random Selection of Data	154

Chapter 8: Structured Programming

Sequence	155
Selection	155
Iteration	157
Invoking Structured Mode	158
STRUCTURED	158
STRUCTURED2	158
CONVENTIONAL	158
USERDEFAULT	159
Restrictions	159
Suggested Coding Standards	159
Sequence Structure	161
PERFORM	161
Selection Structure	162
IF	162
Placement of ENDIF	163
Nesting IF	163
IF Restrictions	164
CASE	165
Nesting and CASE	166
CASE Restrictions	167
EXITCASE	167
Iterative Structure	167
DOWHILE	168
DUNTIL	168
Placement of ENDDO	169
Nesting and DOWHILE or DUNTIL	169
DOWHILE or DUNTIL Restrictions	170
EXITDO	170
ITERATE	171
Automatic Cycle	171
Cycle Without SORT	172
Cycle with SORT	175
Structured Programming Techniques	176
Select Records for SORT	177
ACCEPT	177
REJECT	179
GOTO	179
Structured Programming Examples	179

Chapter 9: USERDEFAULT Mode Programming

USERDEFAULT Option	185
Invoking USERDEFAULT Mode	185

FILE	185
Record Format	186
Record Size	186
Block Size	186
Field Definitions	187
Size/Length	187
Procedure Logic	187
LIST	188
USERDEFAULT Considerations.	188
Examples of Programming in USERDEFAULT Mode	189
The EMPMSTR File	189
OPTION Statement	190
FILE Statement	190
Field Definition Statements	190
LIST Statement	191
SORT Statement	192
CONTROL Statement	194
CONTROL Statement on Multiple Fields.	195
Placement of Data Names on CONTROL and SORT Statements	196
Suppress Keyword	197
Field Definition Statement.	199
Title Statements	200
IF Statement	202
IF Statement with Implied OR	204
IF Statement Using THRU	205
IF Statement with ELSE NEXT ENDIF	206
SORTREC as a Tag Name.	206
SORTREC: as a Reference Point	206
COPY Statement	208
The ARFILE File	211
NOSUM Keyword	214

Chapter 10: Using the MOVE Command

Move Field to Field	219
Move Field to Field—Variable Length	219
Move Literal to Field	220
Move Numeric Literal to a Field	220
Move Character Literal to a Field	220
Move Hexadecimal Literal to a Field	221
Move Blanks to Character Field.	222
Move Field or Literal to a Subtotal Field	222
MOVE NUMERIC	222
MOVE ZONE	223
MOVE with OFFSET	223
COMBINE BITS Using OR	224
AND a Field or Literal with a Field	225
COMBINE BITS Using EXOR	226
MOVE CORRESPONDING Qualifier to Qualifier	227

MOVE UNSIGNED ABSOLUTE NORMALIZED	228

Chapter 11: Using Arithmetic Commands

Valid Arithmetic Operators	231
Indexing	232
Rounding	232
Exponentiation	233
Order of Computation	233
Arithmetic Restrictions	235

Chapter 12: Using the SORT Command

Sorting a File	238
Format 1 Syntax	238
Format 1 Example	238
File Name	238
nK	239
USING dataname	239
USING dataname for Y2K Formats	240
A or D	242
WORK n	242
ON END OF SORTING	244
SORT A FILE ... UNTIL	247
Format 2 Syntax	247
UNTIL	247
UNTIL filename	248
UNTIL dataname	252
Sorting a Data Name ... UNTIL	257
Format 3 Syntax	257
Format 3 Example	257
Record Size	257
USING	258
UNTIL	258
UNTIL filename	259
UNTIL dataname	263
RELSORT and RETSORT Commands	263
Program Narrative	265
Program Listing	265
Sysout	266
Program Narrative	267
Program Listing	267
Limitations and Constraints	268
Problem Areas	269
z/OS	270
VSE	270
z/OS and VSE	270

Sort Considerations	271
VSE	271
z/OS	271
z/OS JCL Requirements	271
VSE JCL Requirements	272
CMS FILEDEF Requirements	273

Chapter 13: Using the CALL Command

Subroutine Name	275
nK.	275
CDLOAD, NOCDLOAD	276
USING	277
Parameters	277
Return Code RETCODE	278
External Subroutine Considerations	278
Language Environment Considerations	279
Database Access	279
COBOL Subroutine Considerations	281
COBOL Problem Areas	284
Assembler Subroutine Considerations	285
PL/I Subroutine Considerations	287
z/OS PL/I Subroutine	287
VSE PL/I Subroutine	287
z/OS PL/I Program with PLIEXIT	288
VSE PL/I Program with PLIEXIT	288
PLIEXIT Source Program	289
Writing a Modify Module	290
PARM	291
SIZE	291
Writing an EXIT Module	291
PARM	292
STATUS	292
Subroutines Link Edited as AMODE(31)	293

Chapter 14: Using Procedural Commands

Example	296
Commands	296
GOTO	296
NEXT	297
ACCEPT	297
REJECT	298
STOP	298
STOPALL	298
QUIT	298
QUITALL	299
PERFORM	299

PERFORM Considerations	300
READ filename.	301
WRITE filename.	301
CLOSE filename.	301
Procedures	302
ON ONE, ENDONE	302
ON ONE Considerations	303
ON END OF INPUT	303
ON END OF INPUT Considerations	305
Indexing	306
Commands Used with Indexing	307
IF	308
MOVE.	308
COMBINE	308
ARITHMETIC	308
PRINT	308
LIST.	308
Indexing Restrictions	308
TRANSLATE	309
Dataname1 FROM	309
Example 1	309
Example 2	309
Example 3	310
INTO dataname2	310
Example.	310
UPPER, LOWER	310
ASCII, EBCDIC	311
TRANSLATE dataname1 [INTO dataname2].	311
Example.	311
EDIT	311
Non-Standard Edits	312
Example 1—Check Protection	312
Example 2—Telephone Number Edit.	312
Standard Edits.	313
Restrictions.	313
Edit Codes	313
Dynamic Allocation	317
Rules for Dynamically Allocating Files	317
JCL Considerations.	318
Procedural Command Examples	320
Example 1—Allocate Facility	320
Example 2—Allocate Facility and MOVE Command in OUTJCL Fields	321
Other Commands Affected by Dynamic Allocation.	322
PICNSAVE Command	322
REPORT Statement	322
LTH Statement	323
CALL Command	323
Diagnostic Output and Error Messages	323

Chapter 15: Using the COPY Commands

Formats	326
COPYX Command—Copies a Member or Book into VISION:Results	326
z/OS COPYx Syntax Examples	328
VSE COPY Syntax Examples	328
\$DEFAULT and \$IF	328
Example 1.	329
Example 2.	329
Examples of Source Code	330
DYAUDREC Contents	330
z/OS Input Stream.	330
Final VISION:Results Program	331
SORTDEF Contents.	331
z/OS Input Stream.	331
Final VISION:Results Program	331
SORTDEF Contents.	332
z/OS Input Stream.	332
Final VISION:Results Program	332
SORTDEF Contents.	332
z/OS Input Stream.	333
Final VISION:Results Program	333
LISTAR Contents	333
z/OS Input Stream.	333
Final VISION:Results Program	334
ARSORT Contents	334
z/OS Input Stream.	335
Final VISION:Results Program	335
ARSORT Contents	335
z/OS Input Stream.	335
Final VISION:Results Program	336
General Considerations	336
Source Format Considerations	337
COPY or COPYE Considerations—z/OS.	337
COPY or COPYE Considerations—VSE	337
COPYC Considerations—VSE	338
COPYL Considerations.	338
CA-Librarian z/OS JCL	338
CA-Librarian VSE JCL	338
COPYN Considerations	338
COPYP Considerations.	339
CA-Panvalet z/OS JCL	339
CA-Panvalet VSE JCL	340
Using COBOL Record Descriptions	340
Instream COBOL Facility	340
Replacing Character Strings.	340
Rules for Using COBOL Record Descriptions.	341
COBOL Examples	342
COPY Member COBOL.	342
Program Listing	342

Instream COBOL	343
COBOL Replacing Literal	343
VISION:Results Statement	343
COBOL Copybook Member	343
Copied Member in VISION:Results	344
COBOL Replacing Pseudo-Text	344
VISION:Results Statement	344
COBOL Copybook Member	344
Copied Member in VISION:Results	344
Prefixing Data Names	344
Using DB2 Tables	345
COPYDB2 Requirements	346
COPYDB2A DD Statement to Override the Default IDs	346
The CALL Attach Implicit Connection Facility	347
TSO Attach Facility	348
IMS Attach Facility	348
VARCHAR Data Type Support	348
Generating Null Indicator Fields	349
COPYDB2 Example	349
JCL DB2 Overrides Example	350

Chapter 16: Using Report Statements

Report Statement	354
z/OS	354
VSE	354
nnn WIDE	355
DELIM	355
nnn LONG	356
nn BETWEEN	356
Modulename ASA	357
Module Name	357
ASA	357
SYS280Rx	357
HTML ddname	357
HTMLTITL Dataname	358
Member Names	360
Naming Conventions	360
Using HTML	361
Creating HTML Documents Using HFS Path	361
Creating HTML Documents Using HTML PDS	364
Dynamic Allocation of HTML Using HFS Path	366
Dynamic Allocation of an HTML Partitioned Data Set	367
HTML Document on the Web Browser PC	369
PLUnnn	369
AND	370

Title and Footing Statements	371
Tn	371
Title Contents	371
FIXED	372
FOOTING	372
WITH n AFTER	372
Additional Information	373
Title or Footing Modification Statement	374
Tn	374
Alphanumeric Literal	375
Numeric Literal	375
Data Names	376
Reserved Words	376
Additional Information	376
Limitations and Constraints	377
LIST Statement	378
LIST Command	379
Data Name Format	380
Keyword	381
SUM	381
SUPPRESS	383
PRIOR	385
SUM1 Through SUM6	386
Data Name	387
dataname AT dataname2	391
dataname AT dataname2 + n	392
Literal	393
TALLY / TALLYn	396
Column Heading	398
Print Line Spacing	400
WITH n BEFORE and WITH EJECT BEFORE	400
WITH n AFTER and WITH EJECT AFTER	401
Alignment of Data Names and Literals at Detail Time (Multi-line)	402
Alignment of Data Names and Literals at ON CHANGE IN and ON FINAL Time	403
Fixed Print Position Reporting	406
Data Name	406
nnn (Print Location)	407
CONTROL or SUBTOTAL Statement	407
ON CHANGE IN	409
Data Name	409
ON FINAL	410
Report Examples	412
Report Print Line Exit—Your Exit Routines	427
Procedure	427
Considerations for Writing Your Report Line Exit	428

Chapter 17: XML Support

Using XML	431
XMLFILE Syntax.	431
XMLFILE ddname	432
XMLWRITE ddname	432
Sample Program	432
Creating XML Documents Using HFS Path	433
Creating XML Documents Using PDSE	434
Creating XML Documents Using Sequential Data Sets	435
Dynamic Allocation of XML Documents Using HFS Path	435
Dynamic Allocation of XML Documents with PDSE.	437
Dynamic Allocation Using Sequential Data Sets	438
XML Data and Schema Document Generation.	439
Sample Program	439
XML Data Document Without Style Sheet	439
XML Data Document with Style Sheet.	440
XML Schema Document	441
XML Documents on PC Web Browser	442
eXtensible Stylesheet Language Transformation (XSLT)	442
Sample XSL Style Sheets	442
XML Data Document Without a Style Sheet.	444
XMLXSL ddname	444
XML HTML Integration	445
XML Data Document with a Link to an HTML Report	445
Sample Program	445
Error Handling	447
WORKAREA Fields	447
XML API Completion and Reason Codes	449
HCSDATA Module Codes	449

Chapter 18: Multiple Reports and Multiple Requests

REPORTnnn Statement	453
SYS280Rx.	454
USE Statement.	454
STATUS dataname	455
PICNSAVE Statement	456
idname.	456
USING dataname.	456
BY nnnnn	457
SYSnnn	457
DISK nnnn	457
TAPE	457
Multiple Requests	458
Freezing and Restoring Multiple Reports	458
Multiple-Report Examples	458
z/OS	459
JCL for Example 1	463

JCL for Example 2	466
VSE	467

Chapter 19: File Print Commands

Printing Using the Option Statement	477
Printing Using the Print Immediate Commands.	478
File Name, Data Name, Indexed Data Name, or Literal	479
LENGTH n or Data Name	480
Other Print Features	482
Comments.	482
Record Number	482

Chapter 20: VSAM Processing

Syntax Format	484
VSAM ESDS FILES	484
VSAM KSDS FILES	484
VSAM RRDS FILES	485
Creating a File	485
Key Sequence VSAM (KSDS) Create	486
Entry Sequence VSAM (ESDS) Create	487
Relative Record VSAM (RRDS) Create	488
Sequential Read	490
Key Sequence VSAM (KSDS) Sequential Read	490
Entry Sequence VSAM (ESDS) Sequential Read	491
Relative Record VSAM (RRDS) Sequential Read	492
Limited (Skip) Sequential Read.	493
Key Sequence VSAM (KSDS) Skip Sequential Read.	493
Generic Search	496
Entry Sequence VSAM (ESDS) Skip Sequential Read	496
Relative Record VSAM (RRDS) Skip Sequential Read.	499
Random Read.	501
Key Sequence VSAM (KSDS) Random Read	501
Entry Sequence VSAM (ESDS) Random Read.	503
Relative Record VSAM (RRDS) Random Read.	505
Sequential Update	507
Key Sequence VSAM (KSDS) Sequential Update.	508
Entry Sequence VSAM (ESDS) Sequential Update	509
Using the RELBYTE or RELBYTEX dataname.	511
Relative Record VSAM (RRDS) Sequential Update.	511
Random Update	513
Key Sequence VSAM (KSDS) Random Update	514
Entry Sequence VSAM (ESDS) Random Update	516
Using the RANDOM/RANDOMX dataname	518
Relative Record VSAM (RRDS) Random Update	519

Sequential Insert and Extend	520
Key Sequence VSAM (KSDS) Sequential Insert/Extend	521
Entry Sequence VSAM (ESDS) Sequential Insert/Extend	522
Relative Record VSAM (RRDS) Sequential Insert/Extend	523
Considerations for Processing Variable-Length Records	524
Record Size	525
Block Size	525
Location of Data Within Record	525
Length of Current Record	525
Output from Itself	525
Output from an Input file	525
File Printing of Variable Files	526
Sorting a Variable File	526
Numeric and Packed Key Considerations	526
KSDS Examples	528
ESDS Examples	531

Chapter 21: ISAM Processing

ISAM.	535
Syntax Format	536
Creating a File	536
ISAM Create	537
z/OS	537
VSE	538
Sequential Read	538
ISAM Sequential Read	539
VSE Considerations	539
Skip Sequential Read	540
ISAM Skip Sequential Read	541
Generic Search	543
VSE Considerations	543
Random Read.	544
ISAM Random Read	544
VSE Considerations	546
Sequential Update	547
ISAM Sequential Update	547
z/OS Considerations.	549
VSE Considerations	549
Random Update	550
ISAM Random Update.	551
VSE Considerations	553
Sequential Insert or Extend	554
ISAM Sequential Extend	554
z/OS Considerations.	555
VSE Considerations	555

Considerations for Processing Variable-Length Records	556
Record Size	556
Block Size	556
Location of Data Within Record	557
Length of Current Record	557
Output from Itself	557
Output from an Input file	557
File Printing of Variable Files	558
Sorting a Variable File.	558
Unblocked ISAM Files in VSE Considerations	558
Sequential or Skip Sequential Read	558
Example	558
Sequential Load or Extend.	559
Random Retrieval.	559
Random Update	560
Numeric and Packed Key Considerations	560
ISAM Examples.	561

Chapter 22: BDAM Processing

Syntax Format	565
Record Addressing Schemes.	566
Relative Track and Actual Key	566
Relative Track and Actual Record Number	566
Relative Block	567
Access Methods	567
Sequential Access	567
Random Access	567
I/O Statistics	567
Random Create.	568
Relative Track Number and Actual Key	568
Relative Track Number and Actual Record Number	570
Relative Block Number	572
Sequential Create	573
Relative Track Number and Actual Key	574
Relative Track Number and Actual Record Number	576
Relative Block Number	578
Random Read.	580
Relative Track Number and Actual Key	581
Relative Track Number and Actual Record Number	583
Relative Block Number	584
Sequential Read	586
VSE	586
z/OS.	586
Relative Track Number and Actual Key	587
Relative Track Number and Actual Record Number	588
Relative Block Number	589

Random Update	590
VSE	590
z/OS	591
Relative Track Number and Actual Key	591
Relative Track Number and Actual Record Number	593
Relative Block Number	596
Other Considerations	597
Processing Variable-Length Records	597
Numeric and Packed Key	599
BDAM Examples	600
Control Totals of Example 1	601
Control Totals of Example 2	602
Output of Example 2	603
Control Totals of Example 3	604
Control Totals of Example 4	608
Control Totals of Example 4	610

Chapter 23: Using PDS and SSL Support

Using the PDS Function	612
Input to VISION:Results	612
FILE	612
PDS Dataname	613
MEMBER Dataname	613
STATUS Dataname	614
FB VB F V U recordsize blocksize	614
LENGTH Dataname	614
COUNT Dataname	614
NEWNAME Dataname	615
INPUT	615
OUTPUT FROM Filename	615
READMEM and READDIR Commands	615
WRITEMEM and WRITEDIR Commands	616
PDS Output Rules and Recommendations	616
ADD Keyword	617
ALIAS Keyword	617
CHANGE Keyword	617
REPLACE Keyword	618
SCRATCH Keyword	619
PDS Examples	619
PDS Directory Format	632
Source Statement Library Support	633
Input to VISION:Results	633
FILE	633
BOOK Dataname	634
STATUS Dataname	634
COUNT Dataname	634
Defining the File	634
Reading a Book	634

SSL Examples	635

Chapter 24: Using MATCH and MERGE

General Flow of Execution Diagram	639
MATCH and MERGE Features	640
Syntax for the MATCH Operation.	641
SETREAD MASTERUP, COMPAREN, and ORIGINAL	642
USING DY282MAT	642
Recommendations and Requirements for Matching	642
Match Conditions in the Procedure Logic	643
SETREAD MASTERUP	643
SETREAD COMPAREN	643
SETREAD ORIGINAL	644
DY282MAT.	644
IF ADVANCED file1	646
IF MATCHED	647
IF MATCHED file1 file2	648
IF DUPLICATE file1	649
IF FIRSTDUP file1.	650
IF LASTDUP file1	651
MATCH Examples	652
MERGE	657
Syntax for the MERGE Operation	657
Recommendations and Requirements for Merging	659
Merge Conditions in the Procedure Logic	659
The Rule for DY282MER	659
Example of Using DY282MER.	660
IF REPLICA	661
Merge Example.	662
Error Analysis During Match and Merge	663
Writing Your Own MATCH Subroutine.	664
DY282MAT Invocation.	664
DY282MAT MATCH Subroutine	665
Generated from Your Special MATCH Logic Statements	667
ON ONE Logic	670
VISION:Results Cycle For MATCH Run Before Your Statements	670
User Statements with Other Special MATCH Logic	671
IF ADVANCED File1	671
IF MATCHED	671
IF MATCHED File1 File2.	671
IF DUPLICATE File1	671
IF FIRSTDUP File1	671
IF LASTDUP File1.	671
Parms Passed to DY282MAT.	672
PARAM1	672
PARAM2	673
PARAM3 through PARAM10	673

Writing Your Own MERGE Subroutine	679
MERGE Subroutine	679
MERGE Logic (Automatic Cycle)	680
ON ONE Logic	681
Cycle for MERGE Run Before Your Statements	681
User Statements with Other Special MERGE Logic	682
IF REPLICA	682
Parms Passed to DY282MER	682
PARM1	682
PARM2	683

Chapter 25: Using Interval Selection

Defining Interval Selection	687
Internal Selection Criteria	688
SAMPLE	688
nn	688
INTERVAL	689
INTERVAL Size Desired	689
Starting Number	689
Maximum Sample Size	689
IF SAMPLING Command	689
IF SAMPLING Command Examples	690

Chapter 26: Using Random Selection

Input to VISION:Results	695
Known Universe	696
SAMPLE	696
nn	696
RANDOM.	696
Population Size	696
Sample Size	697
SEED Original Seed Number	697
Unknown Universe	697
SAMPLE	697
nn	698
RANDOM.	698
Percent (%) of Population	698
SEED Original Seed Number	698
IF SAMPLING Command	699
Random Selection Examples	700
VISION:Results Reserved Words	701

Chapter 27: Table and Array Handling

Allocating Memory	705

Working with Tables	706
Syntax for Defining a Table	706
Table Entry Field Definitions	707
Loading a Table	708
Fixed-Length Entries	708
Variable-Length Entries	708
Sequential Retrieval	709
Fixed-Length Entries	709
Variable-Length Entries	710
Sequential Search	710
Fixed-Length Entries	710
Variable-Length Entries	711
Binary Search	712
Sorting	713
Random Retrieval	713
Fixed-Length Entries	713
Variable-Length Entries	714
Random Replacement	714
Fixed-Length Entries	714
Variable-Length Entries	715
Deleting a Table	715
Physical DELETE	715
Logical Deletion	716
USE Command	717
Error Return Codes	717
Table Examples	718
Working with Arrays	721
Syntax for Defining an Array	721
Loading an Array	722
Retrieving from an Array	723
Deleting an Array	723
USE Command	723
Error Return Codes	724
Array Example	725

Chapter 28: Using Letter Writing

Using the Letter Writing Function	727
Implementing Letter Writing	728
Input to VISION:Results	729
LTH Statements and Keywords	729
LTD Statements and Keywords	733
Additional LTD Statement Information	738
LTD Edit Codes	739
LETTER Command	742
Laser JCL	743
Letter Writing Considerations	744
Channel Skipping with Two-up Letter Processing	745
Optimize and Nooptimize	746

Letter Writing Examples	747
Example 1— Input to VISION:Results	748
Example 1—VISION:Results Output	749
Example 2—Input to VISION:Results	750
Example 2—VISION:Results Output	751
Example 3—Input to VISION:Results	752
Example 3—VISION:Results Output	753

Chapter 29: Using Diagrams and Analysis

Defining Scatter Diagrams, Linear Regression Analysis, and Trend Line Analysis.	767
Scatter Diagram	767
Linear Regression Analysis	768
Correlation Coefficient	770
Coefficient of Determination	770
Coefficient of Non-Determination.	770
Trend Line Analysis	770
Further Sources of Information	770
Input into VISION:Results	771
Regression Command	773
Output from VISION:Results	775
Evaluation of Output	777
Examples of Linear Regression and Trend Line Analysis.	778
Linear Regression Analysis Example	778
Trend Line Analysis Example	780

Appendix A: Reserved Words

Symbols	784
A	784
B	784
C	784
D	784
F	787
G	787
H	787
I	787
J	787
K	788
L	788
M	788
N	788
O	789
P	789
Q	789
R	789
S	789
T	790
U	790

V	790
W	790
X	791
Y	791
Z	791
Numbers	791
Definitions	791

Appendix B: Job Control Language

z/OS JCL	797
VSE JCL	797
VM/CMS	798

Appendix C: z/OS Utilities

JCL Considerations	799
Input File Definitions	800
Output File Definitions	802
Utilities	802
Standard Label Tape to Standard Label Tape	802
Normal	802
Reblocking	803
Variable # of Input Files	803
Variable-Spanned Records with Selection	803
Unlabeled Tape to Standard Label Tape	804
Standard Label Tape to Disk	804
Sequential Data Set	804
ISAM, Blocked	805
ISAM, Unblocked, RKP=0	805
ISAM, Unblocked, RKP Not Equal to 0	806
Standard Label Tape to Card Images	806
Standard Label Tape to Printer	806
Fixed-Length Records, Print Hex and Graphics	806
Variable-Length Records, Print Hex and Graphics	807
Variable-Spanned Records, Print Graphics Only	808
Undefined Length Records, Print Hex and Graphics	808
Partial Records, Print Graphics Only	809
Selected Records	809
Report Tape with ASA Carriage Control	810
Disk to Standard Label Tape	810
Disk (ISAM) to Standard Label Tape	810
Disk (ISAM) to Disk (ISAM)	811
Reorganize ISAM File	811
Disk (Sequential) to Disk	811
Partitioned Data Set—Adding or Replacing a Member in an Existing Library	811
Disk to Disk	812
Undefined Copy	812

Copy Variable-Spanned Records	812
Disk to Card Images.	812
Disk to Printer	813
Print Graphic Only	813
Limited Output, Print Hex and Graphics	813
ISAM, Sequential Read	814
ISAM, Random Read	814
List Members of a Partitioned Data Set.	815
Instream to Standard Label Tape	815
Variable-Length Records.	815
Reformatting Instream Data	816
Instream to Unlabeled Tape.	816
Instream Data Contains JCL	816
Instream to Disk	817
Sequential	817
ISAM, Blocked	817
Unblocked ISAM, RKP=0.	818
Unblocked ISAM with RKP Not Equal to 0	818
Instream to Card Images	818
Instream to Printer	819
VSAM—Define KSDS Fixed Format File Using IDCAMS	820
VSAM KSDS—Tape to Disk.	820
Create/Restore a Fixed KSDS VSAM File from a Fixed Format Sequential File	820
VSAM KSDS—Disk to Tape	821
Backup a Fixed KSDS VSAM File to Another File	821
VSAM KSDS—Disk to Printer	821
Fixed KSDS VSAM File, Print Hex and Graphics	821
Retrieve, Sort in Another Sequence, and Print.	821
VSAM—Define ESDS Fixed Format File Using IDCAMS	822
VSAM ESDS—Tape to Disk.	822
Create/Restore a Fixed ESDS VSAM File from a Fixed Format Sequential File	822
VSAM ESDS—Disk to Tape	823
Backup a Fixed ESDS VSAM File to Another File.	823
VSAM ESDS—Disk to Printer	823
Fixed ESDS VSAM File, Print Hex and Graphics	823
Retrieve, Sort in Another Sequence, and Print.	824

Index

Chapter 1: Introduction

Advantage™ VISION:Results™ for z/OS is an information management and report generator tool with an easy-to-use, English-like programming language that makes it easy to program on any device that allows VISION:Results characters to be input into it.

This guide comprehensively describes the features of VISION:Results, including commands, with their syntax, multiple examples, and a description of all their options.

Consisting of five subsystems, VISION:Results provides:

- Comprehensive information processing features
- Powerful report generation
- Easy document handling
- Convenient utility functions
- Extensive data analysis

VISION:Results supports all IBM file structures and access methods, and is available for z/OS, VSE, and VM/CMS operating systems.

Intended for use by both programmers and non-programmers, VISION:Results accomplishes data processing tasks in a fraction of the time that it would take employing traditional programming languages such as COBOL or PL/I.

Although VISION:Results has the power and flexibility of a full-function language, it is a supplementary tool and should not replace an installation's standard programming language.

For more information about the basic features of VISION:Results, see the *Advantage VISION:Results for z/OS Getting Started*.

Note: Advantage VISION:Results will be referred to as VISION:Results throughout this guide.

Contacting CA Technical Support

For online technical assistance and a complete list of locations, primary service hours, and telephone numbers, contact Technical Support at <http://ca.com/support>.

Chapter 2: Field and Work Area Definitions

Field Definitions

Many of the fields that you reference in your VISION:Results program need to be defined somewhere, either after the FILE statement of the file that the field resides in, or in a work area if these fields are not part of a file. Work area fields that you would be referencing in your program might be constants, arithmetic result fields, or hold fields.

To define a field, typically you would first give it a data name and then follow it with its attributes. The field must be defined prior to its being used in a program. The format of a basic field definition is:

```
dataname field length [starting location]
           [data type] [number of decimals]
           [VALUE nnn | 'xxx' | X'nn' | ALL 'x' |
           ALL X'nn' | NULL | LOWVALUES | HIGHVALUES [REINIT]]
```

Additional items in the definition are discussed later in the chapter. These items relate to how the field appears when it is listed in a report.

The following program illustrates how field definitions are set up and referenced.

```
FILE MSTIN FB 100 1000 STATUS MSTSTAT
  ACCNUM 5 4 NU ACCNAME 25
  BALANCE 5 PD 2 FILLER 22 ACCCODE 1
FILE VSAMDES KSDS RANDOM FINDKEY F 25
  KEYLEN 5 KEYLOC 1 STATUS VSAMSTAT
* RECORD DESCRIPTION
  VSAMNUM 5 VSAMDESC 20

WORKAREA
  COUNTER1 5 PD 2 VALUE 0 REINIT
  COUNTER2 5 PD 2 VALUE 0 REINIT

WORKAREA
  ERRORMSG1 13 1 VALUE 'NO REC FOUND'
  ERRORACCT 5 14
  IF MSTSTAT EQ 'E' STOP ENDIF
  IF ACCCODE EQ 1
    MOVE BALANCE TO COUNTER1 GOTO GETDESC ENDIF
  IF ACCCODE EQ 2
    MOVE BALANCE TO COUNTER2 ENDIF
GETDESC:
  MOVE ACCNUM TO FINDKEY
  READ VSAMDES
  IF VSAMSTAT NE 'Y'
```

Figure 1 WORKAREA Definitions in a Program (Page 1 of 2)

```
MOVE ACCNUM TO ERRORACCT
PRINT ERRORMSG1 LENGTH 18
STOP ENDIF
LIST ACCNUM (ACCOUNT NUMBER) ACCNAME (ACCOUNT'NAME)
VSAMDESC (ACCOUNT'DESCRIPTION) COUNTER1 (CODE 1'BALANCE)
COUNTER2 (CODE 2'BALANCE)
ON FINAL
LIST 'GRAND TOTAL' AT VSAMDESC SUM COUNTER1 SUM COUNTER2
WITH 3 BEFORE
```

Figure 1 WORKAREA Definitions in a Program (Page 2 of 2)

In this example, an account master file is being read and reported on. Additionally, a VSAM file containing a description of each account is read randomly using the Account Number field (ACCNUM) in the master record. The description obtained from the VSAM file is listed in the report along with some of the fields from the account master file. If a record is not found on the VSAM file, an error message is printed and the run is terminated.

The following items make up a field definition:

- [Field Length on page 33](#)
- [Starting Location on page 33](#)
- [Redefining a Field on page 34](#)
- [Data Type on page 34](#)
- [Number of Decimals on page 35](#)
- [VALUE nnn|`xxx'|X`nn'|ALL|NULL|LOWVALUES|HIGHVALUES on page 35](#)
- [REINIT on page 37](#)

Data Name

A data name is required. It can be 2 to 50 characters long (installation default is 50 characters), and it must always begin with an alphabetic character. It can consist of alphanumeric characters and any of the following characters:

#	Pound sign	\	Backslash
\$	Dollar sign	[	Left bracket
%	Percent sign	]	Right bracket
	Solid vertical bar	¢	Cent sign
!	Exclamation mark	_	Underscore
?	Question mark	}	Right brace
<<	Less than sign	{	Left brace
>>	Greater than sign		

Characters not allowed are:

	Blank	;	Semicolon
^	Not sign	:	Colon
&	Ampersand	/	Slash
`	Back apostrophe	,	Comma
~	Tilde	=	Equal sign
	Broken vertical bar	(	Left parenthesis
.	Period	*	Asterisk
)	Right parenthesis	@	At sign
-	Minus sign	+	Plus sign
'	Apostrophe	"	Quotation mark

Field Length

The field length (in bytes) is required. The valid lengths for the various data types are as follows:

- 1 to 32767 for character (CH) data
- 1 to 16 for numeric (NU) data
- 1 to 16 for packed (PD) data
- 1 to 4 for binary (BI) data

In [Figure 1 on page 31](#), ACCNUM is 5 bytes long and ACCNAME is 25 bytes long.

Starting Location

A starting location entry is optional. The number entered as the starting location tells where in the record or work area the field starts. The first byte of a record or work area is location 1.

If no starting location is given and this is the first field entry for the file or work area, it is assumed that the starting location for the field is position 1. Each subsequent field definition then defaults to the starting location immediately after the end of the previously defined field.

In [Figure 1 on page 31](#), the ACCNUM field's location is coded as 4 because it does not start in location 1. If omitted, VISION:Results sets it at 1. You can assign locations to any or all of the fields in the MSTIN file. Looking at the ACCNAME field definition, because only a length, not a location is coded, VISION:Results gives it a starting location of 9, which is the next location in the record after the ACCNUM field.

Note: A data name is located beyond position 32767 in a work area.

FILLER Keyword

If the fields that you are defining are not contiguous and you do not want to calculate starting locations, the keyword FILLER can be used to indicate unreferenced areas between fields to be referenced. Following the keyword FILLER, code the number of bytes between the previously defined field and the next field to be defined.

In [Figure 1 on page 31](#), FILLER is coded between BALANCE and ACCCODE to cause the proper starting location to be assigned to ACCCODE without actually stating it.

The automatic starting location assignment feature of VISION:Results is particularly useful when you are working with COBOL record descriptions that do not contain the starting locations of fields.

Redefining a Field

There are times in a record or work area when you need to define the same locations in several ways. You do this by specifying the starting location of the field that is redefining another.

```
FILE ACCTFLE FB 80 800
NAME 20 11
DATE 6 NU
MONTH 2 31 NU
DAY 2 33 NU
YEAR 2 35 NU
```

The field DATE is being redefined by MONTH, DAY, and YEAR. Because DATE starts in location 31 after NAME, MONTH will start in 31, DAY in 33, and YEAR in 35. For a complete description of redefining fields, see [Work Areas on page 41](#) and [REDEFINE on page 43](#).

Data Type

Specifying the data type is optional. The default is character data type. Use one of the following codes to indicate the type of data contained or to be contained in the field you are defining. The supported codes are:

blank	Character data.
CH	Character data.
NU	Numeric data (zoned decimal).
PD	Packed decimal.
BI	Binary data.

In [Figure 1 on page 31](#), ACCNUM is defined as a numeric field (zoned decimal); ACCNAME is not given a data type, so character is assumed. BALANCE is a packed decimal field.

Number of Decimals

Coding is optional. The default is zero decimals. Code a value in the range of 0 to 9 to indicate the number of digits to the right of the assumed decimal point in the field being defined. Character data cannot be given a number of decimals value.

In [Figure 1](#), BALANCE is defined as having two decimal positions. COUNTER1 and COUNTER2 in the first work area are also two-decimal fields.

VALUE nnn | `xxx' | X`nn' | ALL | NULL | LOWVALUES | HIGHVALUES

When you are defining a field that is in an output record area or a work area, you can have VISION:Results set that field to a particular value. For example, the COUNTER1 and COUNTER2 packed fields defined in the first work area are set to 0 before execution begins. The character field ERRORMSG1 has been assigned a value of NO REC FOUND.

Input file fields cannot be assigned values because the area varies depending on the location of the current record in the file's buffer or buffers. Also, it would be meaningless to set a value because the value is that of the field in the record when it is read. In [Figure 1](#), assigning a value of 0 to BALANCE in the input record would cause an error.

Numeric Values nnn

Numeric values can be expressed with a leading minus sign if negative, commas, and a decimal point. They can contain a maximum of 16 digits. The total length of the numeric literal must not exceed 20 bytes. It can be used in field definitions for numeric (NU), packed (PD), or binary (BI) fields. Zero filling and decimal alignment are automatic (for example, in [Figure 1 on page 31](#), the 2-decimal COUNTER1 field is completely zeroed even though 0 has been given as the value, not 0000000.00).

```
MAXAMOUNT  5  PD  2  VALUE  9,999,999.99
NEG         1  PD      VALUE  -1
ACCOUNT    5  NU      VALUE  421
```

Character Values `xxx'

A character or alphanumeric value, such as ERROR 15 or 317-4275, is expressed by enclosing it in single or double quotation marks. In [Figure 1](#), the ERRORMSG1 field in the work area shows such a value.

A single character value (string of characters) can be up to and including 255 bytes in length. This value can also consist of several lines (broken into multiple literals), each enclosed by single quotation marks. Following is an example, TABLE1, in which the total length is 36 and value (the uppercase alphabet and 10 numbers) is broken into two literals.

```
TABLE1 36 VALUE 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'  
              '0123456789'
```

Above, a 36-byte literal that is used to initialize TABLE1 is being continued to a second line. If the value you are coding in the field definition is smaller than the field it is initializing, the literal is left-aligned and padded with blanks. In the original example, notice that the ERRORMSG1 value is 12 characters but is defined as 13 bytes in length. VISION:Results will move NO REC FOUND to the field.

Hexadecimal Values X`nn'

When you are initializing a field to a value that cannot be expressed using the regular alphanumeric or special character representation, you can use hexadecimal notation. The field to be assigned a value in this manner can be a maximum of nine characters. It must have a character or binary data type and zero decimals. The field and value must be the same length.

```
TABLEX 9 VALUE X'010203040506070809'
```

A table area is being initialized to binary 01, 02, 03, and so on.

```
SWITCH 1 VALUE X'FF'
```

A 1-byte field is being set to high-values (all binary ones).

ALL Values

The keyword ALL propagates a single character throughout a character field. For example, the following statements initialize each field to the same value:

```
FIELDA 5 CH VALUE ALL '*'  
FIELDA 5 CH VALUE '*****'
```

NULL Values

The keyword NULL initializes character fields to blanks and numeric fields to zero. For example, these two statements have the same meaning:

```
FIELD1 5 CH VALUE NULL  
FIELD1 5 CH VALUE ' '
```

These two statements have the same meaning:

```
FIELD2 5 PD VALUE NULL  
FIELD2 5 PD VALUE 0
```

These two statements have the same meaning:

```
FIELD3  4  BI  VALUE NULL
FIELD3  4  BI  VALUE  0
```

These two statements have the same meaning:

```
FIELD4  5  NU  VALUE NULL
FIELD4  5  NU  VALUE  0
```

LOWVALUES Values

The keyword LOWVALUES initializes character fields to binary zeros and binary fields to zero. For example, these two statements have the same meaning:

```
FIELD1  5  CH  VALUE LOWVALUES
FIELD1  5  CH  VALUE X'0000000000'
```

These two statements have the same meaning:

```
FIELD2  4  BI  VALUE LOWVALUES
FIELD2  4  BI  VALUE  0
```

HIGHVALUES Values

The keyword HIGHVALUES initializes character fields to binary X'FF's. For example, these two statements have the same meaning:

```
FIELD1  5  CH  VALUE HIGHVALUES
FIELD1  5  CH  VALUE X'FFFFFFFFF'
```

REINIT

The keyword REINIT coded after your VALUE phrase tells VISION:Results to reset the value at the beginning of each cycle. This is a very useful feature when you want to reset areas to zeros or blanks because it saves you the trouble of doing moves to clear counters, record areas, and so on, before processing a new record or records.

In [Figure 1 on page 31](#), the fields COUNTER1 and COUNTER2 in the work area are set to 0 before each record of the MSTIN file is processed.

REINIT should not be used with fields defined after a SORT statement because the reinitialization process does not occur as expected. When a program contains the SORT command, reinitialization occurs during the release (input phase of SORT only). For more information about the SORT command, see [Chapter 12: Using the SORT Command](#).

Extended Format

The extended format of a field definition looks like this:

```
dataname field length [starting location]
[data type] [number of decimals] [edit code]
[report output field size] [ROUND or ROUNDED]
[column headings] [LEFT or RIGHT] [SUM or NOSUM]
[VALUE nnn | 'xxx' | X'nn' | ALL 'x' | ALL X'nn' | NULL |
LOWVALUES | HIGHVALUES [REINIT]]
```

Field length, starting location, data type, number of decimals, and VALUE are defined in the previous section.

Edit Code

Edit coding is optional. This entry can be coded to override the default edited format of a numeric, packed, or binary field that is being listed in a report. If you do not supply an edit code, the default edit code (P) does not zero suppress or insert commas in the printed field. It edits with a decimal point and a minus sign to the right of the field if negative.

By specifying the edit code, you can cause VISION:Results to place currency signs and negative signs to the left of the number. Placement of negative signs in front of numbers is useful when downloading reports created in VISION:Results to a PC spreadsheet program, most of which require that negative numbers have the minus sign in front of the number.

In [Figure 1 on page 31](#), if the 5-byte numeric field ACCNUM has a value of 00123, it will print as 00123 in the report. If the 5-byte, 2-decimal field COUNTER1 contains a value of 999999999, it prints as 009999999.99 (two digits were added because the field is being summed).

For a complete list of VISION:Results edit codes, see [Edit Codes on page 313](#).

Report Output Field Size

This entry is optional. It indicates how many whole number positions and how many decimal positions are to be printed on the report if the field is printed (not applicable to character data fields). The format for this entry is x.y, where x represents the number of digits to the left of the decimal point and y represents the number of digits to the right of the decimal point. The format can also be x only, or .y if only the number of decimal positions is being changed.

In [Figure 1 on page 31](#), COUNTER1 is defined:

```
COUNTER1 5 PD 2 VALUE 0 REINIT
```

A 5-byte packed field contains nine digits [(2 x 5)-1]. Because two decimal places are specified in this example, the field contains seven integer positions and two decimal places with the maximum value possible of 9,999,999.99.

Because the field is being totaled (see SUM COUNTER1 in the ON FINAL LIST statement), VISION:Results adds two integer positions so that up to nine integer positions and two decimal places print with a maximum value possible of 999,999,999.99. If you only want to print six integer positions and no decimals, you can code the following in your field definition:

```
COUNTER1 5 PD 2 6.0 VALUE 0 REINIT
```

You cannot indicate only 6. Doing so means 6.2, because the default number of decimal positions to print is 2, the same as the number of decimal positions in the field.

CA strongly recommends that the use of the report output field size be carefully considered for fields defined with edit codes G, J, K, or L. An insufficient size causes truncation of the minus sign and can result in extraneous data being printed on the report.

If the output print size entry is omitted, VISION:Results calculates the size in the following manner:

- If the field is not totaled (SUM is not used with the field in a LIST statement), the number of digits to print is the number of digits in the field. For example, a 4-byte packed field contains seven digits; so if you are printing a 4-byte packed, 2-decimal field, VISION:Results assigns five digits to the left of the decimal point and two digits to the right for a maximum printable value of 99,999.99.
- If the field is totaled (SUM is used with the field in a LIST statement), VISION:Results assigns two more digits to the left of the decimal point when printing to allow for more significance. In the case of a totaled 4-byte, 2-decimal packed field, VISION:Results provides for seven digits to print to the left of the decimal point and two digits to the right for a maximum printable value of 9,999,999.99.
- The number of decimal positions printed in the report defaults to the number in the field.

ROUND or ROUNDED

The keywords ROUND or ROUNDED are optional. Use either of these keywords when the number of decimals for the field on output is less than the number of decimals for that field on input, and you want the value to be rounded before it is truncated. Otherwise, only the truncation occurs.

In [Figure 1 on page 31](#), if COUNTER1 is defined as follows, the number of decimal positions is truncated to 0 when the field is printed. With the ROUNDED option coded, if COUNTER1 has the value of 1234.56, it prints as 1,235.

```
COUNTER1 5 PD 2 E 7.0 ROUNDED VALUE 0 REINIT
```

Column Heading

In a VISION:Results report, the default column heading of a field is the data name of that field unless overridden in the field definition or in the LIST statement. In the original example, the LIST statement column headings are stated for every field. The field ACCNUM is given the heading (ACCOUNT NUMBER), ACCNAME is given the heading (ACCOUNT'NAME), and so on. This can also be done during the definition of the field:

```
ACCNUM 5 4 NU (ACCOUNT NUMBER)
ACCNAME 25 (ACCOUNT'NAME)
COUNTER1 5 PD 2 (CODE 1'BALANCE) VALUE 0 REINIT
```

It is preferable in most cases to indicate the headings in the LIST statement. This allows you to keep them along with the other field attributes in a library and COPY them into the program, rather than coding them each time.

Note: A LIST statement column heading takes precedence over one in the field definition.

The apostrophe in the middle of the column heading is used to stack the column heading. For example, (ACCOUNT'NAME) is shown as:

```
ACCOUNT
NAME
```

A maximum of nine column headings can be stacked. A maximum of 30 characters is allowed for column headings.

LEFT or RIGHT

The keywords LEFT or RIGHT are optional. Use them to indicate that the column heading is to be aligned to the LEFT or RIGHT of the column instead of being centered above it on the report. The keyword applies if the default column heading (the data name) is used or if the user specifies a column heading as part of the data name definition or the actual LIST statement.

```
NAME 25 85 LEFT
BALANCE 5 PD 2 (BALANCE'OWING) RIGHT
```

SUM or NOSUM

The keywords SUM or NOSUM are optional. They have meaning only if VISION:Results is running in OPTION USERDEFAULT mode. (For a complete discussion of the USERDEFAULT option, see [Chapter 9: USERDEFAULT Mode Programming](#).) In the other modes, VISION:Results ignores the keywords. In OPTION USERDEFAULT mode, VISION:Results automatically totals all non-character fields specified on a LIST statement. You can specify NOSUM on the data name definition if you do not want the field totaled.

The keyword SUM on the data name definition indicates that the field is to be totaled. This keyword is not necessary because this is the default for all non-character fields. The keywords SUM and NOSUM are ignored if they are specified for a character field.

```
TRANS 7 NU NOSUM
BALANCE 5 PD 2 SUM
```

A data name with the same name as the file is automatically generated by VISION:Results. It is a 1-byte character field whose location is the first byte of data in the record.

Work Areas

Work areas are fields or areas you can set up in your program to be used however you want. You can use them to hold some counters or a table area or a record. This is done by first coding the command WORKAREA and following it with field definitions.

```
WORKAREA
COUNTER1 5 PD 2 VALUE 0
COUNTER2 5 PD 2 E 7.0 ROUNDED VALUE 0
SWITCH 1 BI VALUE 0 REINIT
ERRORMSG 15 VALUE 'NO RECORD FOUND'
TABLE1 2000
RECHOLD 80
```

Work areas can be named only if OPTION QLF is in effect. You can name a work area by coding:

```
WORKAREA NAMED workareaname
```

The work area name can be from one to eight characters and must conform to the naming conventions for files. (For a complete description of the FILE command, see [Chapter 6: Using the FILE Command](#).) By naming a work area, you can reference fields within it that have names which are also defined elsewhere in your program. For example:

```
OPTION QLF
.
.
WORKAREA NAMED WORK5
DATE 6
MONTH 2 1 NU
DAY 2 NU
YEAR 2 NU

MOVE 100484 TO WORK5.DATE
```

Work area fields are defined in the same way as fields defined following FILE statements. They can have size, location, data type, and number of decimal attributes. They can be assigned values as well.

If you specify the keyword QLF or QUALIFIERS on the OPTION statement in your program, you can also use the qualified data name feature. This means that you can use non-unique data names for your work area field definitions. When you reference those fields in your program, the work area name is the qualifier. (For details about OPTION QLF, see [MOVE CORRESPONDING and OPTION QLF on page 96.](#))

```
OPTION QUALIFIERS
.
WORKAREA NAMED DATE
  MONTH 2 NU
  DAY    2 NU
  YEAR   2 NU

WORKAREA NAMED QDATE
  MONTH 2 NU
  DAY    2 NU
  YEAR   2 NU

MOVE 10 TO DATE.MONTH
MOVE 12 TO QDATE.MONTH
```

In this example, the non-unique data names MONTH, DAY, and YEAR are being used. These data names must always be qualified with the named work area name.

Do not use a data name that is defined under another data name. For example, the following would be invalid because the data name MONTH is defined under the DATE and QDATE work areas.

```
MOVE 10 TO MONTH
```

When you name a work area, VISION:Results automatically generates a data name with the same name as the named work area. It is a 1-byte character field whose location is the first byte of the work area.

You can place work area coding almost anywhere in your program, but the fields must be defined before they can be referenced. The best place to put your work area definitions is immediately after the FILE statements and their field definitions.

VISION:Results automatically allocates the work area and initializes the fields you have defined for it with the proper values before your program begins execution. If you do not assign values, the fields are blank. The starting location of a work area is 1. You can assign locations if you like. This is sometimes necessary when you are redefining a field. The maximum length of a work area definition is 32767. The maximum total size of all work areas is 288K — 32K for each output file.

```
WORKAREA
  DATE 6      NU
  MO   2  1  NU
  DA   2  3  NU
  YR   2  5  NU
```

The DATE field is redefined and broken down into month, day, and year fields. The starting location of MO is 1, DA is 3, and YR is 5. You can omit the starting locations for DA and YR. VISION:Results assigns them those locations anyway.

Careful assignment of starting locations can prevent problems with successive fields being improperly assigned started positions.

The following examples show how you can assign the field COUNTER1 with a starting location of 5, which is also the last 2 bytes of the DATE field.

```
WORKAREA
DATE      6      NU
YR        2  1  NU
MD        2  3  NU
DA        2  5  NU
COUNTER1  2  5  PD  VALUE 0
```

Preferred method:

```
WORKAREA
DATE      6      NU
REDEFINE AT DATE
YR        2      NU
MO        2      NU
DA        2      NU
REDEFINE AT DA
COUNTER1  2  PD  VALUE 0
```

You can have multiple work areas. The advantages of doing this are:

- Being able to define work area fields anywhere in your program.
- Causing the starting location for fields in a work area to reset to 1. (The first field in a work area has the starting location of 1 assigned.)
- Allowing work area coding to be stored in libraries and copied in wherever they are needed in a program.
- Ensuring that your field is aligned on a doubleword boundary. (Each work area begins on a doubleword.)

REDEFINE

This keyword allows you to supply additional definitions of previously defined fields or areas.

The REDEFINE statement looks like this:

```
[REDEFINE|REDEF] [AT] dataname1
```

where dataname1 is any field already defined in the current FILE or work area.

REDEFINE is normally used in conjunction with a dataname1 of which exact location within the record or work area is not known.

```
WORKAREA
.
.
DATEWK 6
REDEFINE AT DATEWK
MONTHW 2
DAYW 2
```

```
YEARW 2  
.  
.
```

When you do not specify a location in a field definition, that field is assigned a location immediately adjacent to the previous field. REDEFINE causes the default next location to be reset to the location of dataname1. In the example above, MONTHW is assigned the same location as DATEWK, DAYW is 2 beyond that, and so on.

To define a record that has variable-length trailers, code the following:

```
FILE ACCTMST  
  ACCTNO 4 PD  
  ACCTNAME 20  
  ACCTTYP 2  
.  
TRAILERID 1  
  TR1FLD1 5 NU 2  
  TR1FLD2 3 PD  
  TR1FLD3 6  
.  
REDEF AT TR1FLD1  
  TR2FLD1 20  
  FILLER 5  
  TR2FLD3 7 PD 5  
.  
RDEF AT TR1FLD1  
  FILLER 18  
  TR3FLD4 6 NU  
.  
.  
.
```

Note: You can reposition at a field other than the one just defined, and you can REDEFINE an area as many times as necessary.

Special Fields and Reserved Words

VISION:Results sets up a useful group of special fields. For example, a field called DYLDATE contains the current date in MM/DD/YY format. You can reference this field in your program statements:

```
T1 'ACCOUNT NUMBER REPORT'  
T1+90 DYLDATE
```

The current date is being put into the title of the report.

```
MOVE DYLDATE TO DATHOLD
```

The current date is being moved to another field. These special fields have been given size, data type, and decimal attributes, the same as if you had defined them in a FILE statement or work area. The DYLDATE field is 8 bytes in size and is a character (CH) data type. Another field, DYLCOUNT1, which is the first of a series of 126 counters (DYLCOUNT1 through DYLCOUNT99, DYLCOUNT00, DYLCOUNT0A through DYLCOUNT0Z), has a size of 10, a packed data type (PD), and decimal positions of 0. Additionally, if it is to be listed in a report, the edit code attribute default is A.

You can override certain attributes of these fields by coding a pseudo definition in a work area. For example, if you want to override the 0 decimal position attribute of DYLCOUNT1, do the following:

```
WORKAREA
  DYLCOUNT1 2
```

If you think that you have just set the size to 2, this is not the case. These special fields cannot have their size, location, or data type changed, so the 2 being coded in this field definition means the number of decimal positions.

```
WORKAREA
  DYLCOUNT1 2 E (MY COUNTER)
  .
  .
  .
LIST ACCOUNTNUM ACCOUNTCDE DYLCOUNT1
```

The value in DYLCOUNT1, when listed in the report, is edited in the E format and the column heading is MY COUNTER.

BLANK or BLANKS

Size 1, type CH. The value is blank. This field can be used to space fill any size character field because after a value is moved into a character field, it is space filled to the right.

```
MOVE BLANKS TO NAME
```

This field can also be used to compare against a character field.

```
IF NAME EQ BLANKS GOTO GETNAME ENDIF
```

DYLCENTRY1

Size 2, type CH. The value of DYLCENTRY1 is assigned by the DYLINSTL parameter CENTRY1 or set to a specific value by MOVE logic in your program. Use DYLCENTRY1 to override the DYLINSTL parameters. Values can be any numeric value from 00 to 99. If no parameters are set by the DYLINSTL macro or by DYLCENTRY1, VISION:Results defaults to blanks. For details on the DYLINSTL parameters, see the *Advantage VISION:Results for z/OS Installation Guide*.

```
MOVE '19' TO DYLCENTRY1
```

DYLCENTRY1 sets up the century number of the output year for the Excel subroutines DYLDATE, DYLBETDT, DYLFMTJG, and the input year in the “from date” field for DYLSELDT.

DYLCENTRY2

Size 2, type CH. The value of DYLCENTRY2 is assigned by the DYLINSTL parameter CENTRY2 or set to a specific value by MOVE logic in your program. Use DYLCENTRY2 to override the DYLINSTL parameters. Values can be any numeric value from 00 to 99. If no parameters are set by the DYLINSTL macro or by DYLCENTRY2, VISION:Results defaults to blanks. For details on the DYLINSTL parameters, see the *Advantage VISION:Results for z/OS Installation Guide*.

```
MOVE '20' TO DYLCENTRY2
```

DYLCENTRY2 sets up the century year number of the input “to date” year for the DYLSELDT and DYLBETDT Excel subroutines. This allows you to enter a two-digit year yet receive a four-digit output.

DYLCNAME1 Through DYLCNAME99, DYLCNAME00, DYLCNAME0A Through DYLCNAME0Z

Size 20, type CH, and with the initial value of blanks. One hundred twenty-six 20-byte label fields that you can set to identify the corresponding DYLCOUNTx counters when they are printed on the VISION:Results end-of-run statistics listing. If you do not move your own labels into these fields, default labels of TOTAL NUMBER 01, TOTAL NUMBER 02, and so on, are printed to the left of the counters.

```
ON ONE
  MOVE 'RECORD CODE A' TO DYLCNAME1
  MOVE 'RECORD CODE B' TO DYLCNAME2
  MOVE 'RECORD CODE C' TO DYLCNAME3
ENDONE
IF RECODE EQ 'A'
  DYLCOUNT1=DYLCOUNT1+1 GOTO WRTOUT ENDIF
IF RECODE EQ 'B'
  DYLCOUNT2=DYLCOUNT2+1 GOTO WRTOUT ENDIF
IF RECODE EQ 'C'
  DYLCOUNT3=DYLCOUNT3+1 GOTO WRTOUT ENDIF
.
```

Your labels are set for the three counters being incremented.

DYLCOMMON

Size 4, type BI. The value is a memory address. This field contains the address of the VISION:Results common area, and it is used as a parameter that is passed by a CALL statement to a user module. You might need this if you have to gain access to VISION:Results DTF/DCBs, for example.

DYLCOMRG

Size 11, type CH. The value is used for multiple purposes. It is a copy of bytes 12-22 of the VSE partition communications region. It is used to communicate the table, array, or XML API completion code and reason code. It contains the value of register 15 for the CSV routines, TOCSV and FROMCSV. Changing the value in DYLCOMRG has no effect on the actual value in the communications region.

DYLCOUNT1 Through DYLCOUNT99, DYLCOUNT00, DYLCOUNT0A Through DYLCOUNT0Z

Size 10, type PD, decimals 0, edit code A if listed in a report, and initial value is 0. One hundred twenty-six 10-byte packed counters that you can use to maintain your own totals independent of report writer totaling. If you reference any of these counters, the series is printed as part of the end-of-run statistics.

```
IF RETYPE EQ 'A'
  DYLCOUNT1=DYLCOUNT1+1 GOTO WRTOUT ENDIF
IF RETYPE EQ 'B'
  DYLCOUNT2=DYLCOUNT2+1 GOTO WRTOUT ENDIF
IF RETYPE EQ 'C'
  DYLCOUNT3=DYLCOUNT3+1 GOTO WRTOUT ENDIF
.
.
.
```

Counts of record types A, B, and C are being maintained in the counters. These counts are printed in the end-of-run statistics.

DYLDATE

Size 8, type CH. The initial value is the current date in the format MM/DD/YY. If the DYLINSTL parameter EURODAT was selected, the format is DD/MM/YY.

```
T1 'REPORT'
T1+1 DYLDATE
```

The current date is to be placed in title 1, location 1.

DYLDATE4

Size 10, type CH. The initial value is the current date in MM/DD/YYYY format. If the DYLINSTL parameter EURODAT was selected, the format is DD/MM/YYYY.

DYLDATEPAG

Size 30, type CH. The value is the current date and page number edited and ready for insertion in report titles.

The format is DATE**yy**MM/DD/YY**yy**PAGE**yy**NNNNNNNN.

DYLDATEPG4

Size 32, type CH. The value is a four-digit current date and page number edited and ready for insertion in report titles.

The format is DATE**mm**/DD/YYYY**mm**PAGE**mm**NNNNNNNN.

DYDLA

Size 4, type BI. The value is a memory address. This field is always passed as the first parameter in a CALL statement when invoking DYLTDLI (IMS/DL/I database interface option). DYDLA should never be modified in your program.

DYLETIME

Size 8, type CH. The value is the VISION:Results execution start time in the format HH.MM.SS. The format can be changed with the OPTION TIMECOLON statement or by the DYLINSTL parameter TIMESEP. For details, see the *Advantage VISION:Results for z/OS Installation Guide*.

```
T1 'REPORT'  
T1+90 DYLETIME
```

Above, the execution start time is placed in title 1 of a report.

DYLILDATE

Size 4, type BI. The value is the current date in Lilian format. This is the number of elapsed days since October 15, 1582 – when the Gregorian calendar was established.

DYLGREG

Size 4, type PD. The value is the current Gregorian date in packed format. Packed format is 0MMDDYYs. If the DYLINSTL parameter EURODAT was selected, the format is 0DDMMYYs. When used in a LIST or TITLE statement, the format is DD/MM/YY in an 8-byte character format.

DYLGREG4

Size 5, type PD. The value is the current Gregorian date in the four-digit year format MM/DD/YYYY. When used in a LIST or TITLE statement, the format is MM/DD/YYYY in a 10-byte character format. If the DYLINSTL parameter EURODAT was selected, the format is DD/MM/YYYY.

DYJULIAN

Size 3, type PD. The value is the current Julian date in packed format. The format is YYDDDs.

```
IF EXPIRDATE LT DYJULIAN GOTO LISTDET ENDIF
```

In this example, an expiration date in Julian format is compared to the current Julian date.

DYJULIAN4

Size 4, type PD. The value is the current Julian date in the four-digit year format YYYYDDD.

DYLLINE

Size 2, type PD. In this field, VISION:Results maintains the packed line count of the current report page.

```
MOVE 60 to DYLLINE
```

A high line count is being placed in the DYLLINE field to force a page eject.

Also note that, if neither title, footing statements, nor column headings are supplied, automatic page ejection (based on the number of lines per page) does not occur. To override this without using title or footing statements or column headings, you can supply a title/footing statement with no title contents:

```
T1
```

DYLNOTOT

Size 1, type CH. The initial value is blank. This is a control total inhibit switch that you can set before a detail line 1 LIST statement is issued. Setting this switch to 1 causes the report writer not to add any summed field values to their counters. DYLNOTOT is reset to blank before leaving the report writer.

```
IF RETYPE EQ 'A'
  MOVE 1 TO DYLNOTOT ENDIF
LIST ACCNUM ACCAMT1 ACCAMT2
ACCEPT
ON CHANGE IN ACCNUM
  LIST ACCNUM SUM ACCAMT1 SUM ACCAMT2 'ACCOUNT TOTAL'
ON FINAL
  LIST SUM ACCAMT1 SUM ACCAMT2
```

Before the detail line is listed, a check is made to see if a field in the record has the code of A. If so, the DYLNOTOT switch is set to 1. This keeps the report writer from adding the values in ACCAMT1 and ACCAMT2 to their respective total counters. When the level 1 break occurs (ON CHANGE IN ACCNUM), the subtotals reflect all listed records except those with the A code. The same thing happens when final totals (ON FINAL) are listed.

DYLNRP

Size 1, type CH. The initial value is blank. DYLNRP is a detail line inhibit switch that you can set before the first detail line LIST statement is issued. Setting this switch to 2 causes the report writer to inhibit the printing of detail lines.

```
CONTROL ACCOUNT

SORT ARFILE USING ACCOUNT
MOVE 2 TO DYLNRP
LIST SUPPRESS ACCOUNT BALANCE

ON CHANGE IN ACCOUNT
  LIST ACCOUNT SUM BALANCE '***TOTAL**'
```

In this example, a 2 is moved to DYLNRP before the first detail line LIST statement. This tells the report writer to inhibit the printing of all detail lines. Only the summary LIST statements are executed. In this example, only the ON CHANGE IN LIST statement is executed.

DYLONG

Size 1, type PD. The value is 1. This is useful for counting.

DYLPAGE

Size 13, type CH. The value is PAGE followed by the edited page number. The format is PAGE000000000000. This is useful for placing the report page number in report titles.

DYLPAGE1 Through DYLPAGE7

Size 1 through 7, type CH. DYLPAGE1 is an edited 1-character page number. DYLPAGE2 is an edited 2-character page number, and so on. This is useful for including any size page number in your report.

```
T1 'REPORT'
T1+130 DYLPAGE2
```

A 2-digit page number is printed in locations 130 and 131 of title 1.

DYLPARM

Size 60, type CH. This field holds variable data provided by either the PARM parameter in the EXEC statement (z/OS only) or the DATA xxxx phrase in the VISION:Results OPTION statement.

```
//S001 EXEC PGM=DYL280,
// PARM='UA=011596,01,02'
.
.
WORKAREA
  PARHOLD 12
  ACCTNUM 6 1
  COMMA1 1
```

```
NUM1 2 NU
COMMA2 1
NUM2 2 NU
ON ONE
MOVE DYLPARM TO PARHOLD
ENDONE
.
.
```

In a z/OS run, variable data is being introduced by the PARM parameter in the EXEC statement. To code this, following the PARM=, a 'UA=' is coded preceding the data to be passed to the program. The variable data is terminated by another single quotation mark. When program logic is entered for the first time, the variable data resides in the DYLPARM field. The program does a one-time only move of DYLPARM to PARHOLD. This is being done in order to separate the variable data into individual fields (ACCTNUM, NUM1, and NUM2) so they can be referenced. A maximum of 58 bytes can be coded.

```
OPTION DATA '011596,01,02'
.
.
.
MOVE DYLPARM TO PARHOLD
```

This can be done in either a z/OS or a VSE run to introduce the same variable data as in the preceding z/OS example. A maximum of 38 characters can be coded.

In a z/OS run, if you use both methods in the same program, the PARM value in the EXEC statement overrides the OPTION DATA value.

DYLPARMLLEN

Size 3, type PD. This field contains the length of the data stored in DYLPARM.

```
//S001 EXEC PGM=DYL280,PARM='UA=011596,01,02'
      DYLPARMLLEN now contains a value of 12.
or
```

```
OPTION DATA '123456789'
      DYLPARMLLEN now contains a value of 9.
```

DYLPAGE

Size 6, type PD. The value is the current report page number. This field is incremented each time a new page is started. You can reset this field.

DYLPROGRAM

Size 8, type CH. DYLPROGRAM is a reserved word used to specify a program identification name. A user-specified program identification name may be moved into DYLPROGRAM. If DYLPROGRAM is not specified by the user, the default is the library member name allocated to the SYSIN DD statement. If the SYSIN is an instream data set, 'INSTREAM' is assigned to the program identification name. If SYSIN is allocated to a sequential data set, a program identification name is not assigned. A MOVE statement should be used to populate DYLPROGRAM with a valid program ID name in these two cases if it is referenced in the program.

The program identification name is determined and set either during the compile phase of a compile/go program or during the freeze phase of a freeze/restore program. The listing or the output of a restore program displays the program identification name set during the freeze. An example is shown in [Figure 2](#). Examples of the report output are shown in [Figure 3](#) and [Figure 4](#).

```
JCL
//SYSIN      DD  DSN=YOUR.RESULTS.PROGLIB(ARSAMPLE),DISP=SHR
Source program:
REPORT
FILE ARFILE FB 352
COPY ARFIELDS
ON ONE
    PRINT '*****'
    PRINT '** VISION:RESULTS START BANNER'
    PRINT '*****'
    PRINT '** PROGRAM      : ' DYLPROGRAM
    PRINT '** RUN DATE     : ' DYLDATE
    PRINT '** RUN TIME     : ' DYLETIME
    PRINT '*****'
ENDONE
LIST ACCOUNT NAME BALANCE
T1  'ACCOUNT RECEIVABLES'
T1+1 'PROGRAM ID: '
T1+13 DYLPROGRAM
T1+60 DYLDATEPAG
FIN
```

Figure 2 DYLPROGRAM Example

```
*****
** VISION:RESULTS START BANNER
*****
** PROGRAM      : ARSAMPLE
** RUN DATE     : 01/07/06
** RUN TIME     : 18.55.20
*****
```

Figure 3 DYLPROGRAM Report Output: Banner Page

PROGRAM ID: ARSAMPLE	ACCOUNT RECEIVABLES	DATE 01/07/06	PAGE 1
ACCOUNT			
NUMBER	NAME	BALANCE	
8006547	TORRES, ERNESTO	44.99	
6208657	CHO PYUNG, SUH	32.00	
	.		
	.		
	.		

Figure 4 DYLPROGRAM Report Output: Page 1

DYLPRTCOMM

Size 10, type CH. The initial value is blanks. This field can be set to hold an identifier to print in a comments area to the left of a record or field being printed by the print immediate (PRINT, HEXPRINT, and so on) commands. If you do not move anything in before printing, the file name or data name appears.

```
MOVE 'MASTER' TO DYLPRTCOMM
PRINT MSTFLE
```

The identifier MASTER prints to the left of the MSTFLE record being printed. If you had not moved anything to DYLPRTCOMM, the file name MSTFLE would have printed.

If there is more than one data name in your print command, you must move the data names to DYLPRTCOMM before issuing your print command in order to have them appear in the comments area.

DYLPRTMAXL

Size 2, type PD. The value is the maximum lines per page to be allowed for file printing. It has been set either by default or by the LONG keyword of the OPTION statement. You can vary this during execution.

DYLPRTNUMB

Size 4, type PD. The value is the count of records file printed by VISION:Results. You can examine this field in the logic to terminate the run after a certain number of records have been printed.

DYLPRTPAGE

Size 3, type PD. The value is the page number for the file print listing. VISION:Results maintains this field. You can examine this field in the logic to terminate the run after a certain number of pages have been printed.

DYLREPMAXL

Size 2, type PD. The value is the maximum lines per page to be allowed for the report. It has been set either by default or by the LONG keyword of the REPORT statement. You can vary this during execution.

DYLRESET

Size 1, type BI. The initial value is 0. You can set this switch to reset the page number to 1 on the report. If you move a 1 to DYLRESET, after a level 1 control break occurs, the next page number is 1. If you move a 2 to DYLRESET, after a level 2 control break occurs, the next page number is 1. You can do this up to and including a level 6 control break. This switch is not reset to 0 by VISION:Results.

```
CONTROL ACCNUM
.
.
.
ON CHANGE IN ACCNUM
  MOVE 1 TO DYLRESET
  LIST ACCNUM SUM ACCAMT1 'ACCOUNT SUBTOTAL'
    WITH 2 BEFORE AND EJECT AFTER
.
.
.
T1 'ACCOUNT REPORT' DYLPAGE
```

A 1 is moved to the reset switch to indicate that after the level 1 control break (ON CHANGE IN ACCNUM) occurs and the control break line is printed, the next page printed has the page number of 1 (that is, if you are printing one of the VISION:Results special page number fields in the title). Specify EJECT AFTER in the control break's LIST statement when using DYLRESET.

DYLRETURN

Size 2, type BI. The initial value is a return code of zero. You can set this before your run terminates to cause either a condition code to be passed to the next step, or an abend to occur (for VSE, the system must be at Release 2.1 or greater and you must specify DYLINSTL parameter R15RC). The setting of DYLRETURN does not cause the run to terminate. Processing continues until all files are at end or you execute a STOP command. The values you set have the following result:

0	Normal completion.
1-99	Condition code value to be passed.
100-4095	Abend with system dump when processing terminates. VSE abends with an ILLEGAL SVC 32 and the return code value in register 0. z/OS abends with a user completion code that is the same as what has been set in DYLRETURN.

```
.  
. .  
IF STATUSFLG NE 'Y'  
  GOTO ERROR2 ENDIF  
. .  
ERROR2:  
  MOVE 502 TO DYLRETURN  
  STOP
```

If an error is detected, the ERROR2 logic moves the code of 502 to DYLRETURN and stops the run. Any report is finished, files are closed, and run information is printed before the abend occurs.

DYLSYSDATE

VSE only. Size 9, type NU. DYLSYSDATE contains the system date from the partition communication region (offset X'4F'). The first 6 bytes contain the Gregorian date (MMDDYY or DDMMYY), and the last 3 bytes contain the Julian day number (DDD). This date is not affected by the // DATE statement.

DYLSYSPARM

z/OS: DYLSYSPARM contains the PCB address list reflected in the PSB. For more information, see [Database Access on page 279](#).

VSE: Size 8, type CH. DYLSYSPARM contains 0 to 8 bytes of data from the SYSPARM=statement on your EXEC statement, padded on the right with blanks. This is release-dependent.

DYLTIME

Size 4, type PD. The edit code is A if listed in the report. The value is the VISION:Results execution start time. Packed format is 0HHMMSSs.

DYLTRAN

Size 1, type CH. DYLTRAN refers to the first byte of a 256-byte translate table. If you do not want to use the FROM and TO option of the TRANSLATE command, you can specify your own translate table by moving values to the appropriate offset from DYLTRAN. Use indexing to do this.

You want to change all '.' (X'4B') to blanks. You must make the change to the translate table.

```
MOVE 75 TO INX
MOVE ' ' TO DYLTRAN (INX)
TRANSLATE FIELDABC
```

This results in all occurrences of '.' in FIELDABC being translated to X'40.'

DYLUPSI

Size 1, type BI. The value is a copy of the VSE UPSI byte. Changing this value has no effect on the actual UPSI byte.

GETCOM dataname

VSE only. This statement causes from 1 to 256 bytes to be moved from the JCA to dataname. The number of bytes moved depends on the length of dataname. The data is moved from left to right.

PUTCOM dataname

VSE only. This statement causes from 1 to 256 bytes to be moved from dataname to the JCA. The number of bytes moved depends on the length of dataname. The data is moved from left to right.

SPACE or SPACES

Same as BLANK or BLANKS.

Chapter 3: Syntax Rules

This chapter explains the syntax rules for VISION:Results.

Using Punctuation

Do not use punctuation, such as colons (:) or periods (.), to delimit command statements, parameter lists, or a list string in an IF statement.

```
MOVE FIELDA TO FIELDB  
CALL SUBRTN USING PARM1, PARM2, PARM3  
IF ACCOUNT EQ 'NA', 'MO', 'PO' THRU  
'WO' GOTO CHECK ENDIF
```

Figure 5 Example of an IF Statement

The exception is when you are expressing numeric literals that have commas, decimal point and negative amount indications, and reference point tag names that must be followed by a colon (:).

```
573,624          8,973,361.88  
591,324.08       -12  
-.01            -637,128.19  
LISTDET:        WRT:
```

Figure 6 Example of the Use of Colons

Asterisks (*) and semicolons (;) are used for comments. The asterisk (*) can be used only if placed in position 1 (byte 1) of the input record, regardless of whether the OPTION COLUMNS override is used. The asterisk is also used as the arithmetic symbol for multiplication. Semicolons can appear anywhere (1 to 80) in the free-form input record.

```
*COMMENTS REMARKS  
;COMMENTS  
MOVE FIELDA TO FIELDB ;DATE TRANS
```

Figure 7 Example of the Use of Asterisks and Semicolons

Continuation Rules

VISION:Results has a free-form syntax in which commands may flow from one line to the next. A line break may come at any place a space would separate data names, field names, or keywords. Literals may be continued on another line only in TITLE statements and on the VALUE keyword in field definitions. When continuing literals used on TITLE statements or VALUE keywords, the literal is delimited with quotation marks. Literals in procedure logic and LIST statements cannot be continued onto separate lines.

```
MOVE FIELDA TO
FIELDB

LIST BALANCE ACCOUNT TRANS
WITH 2 AFTER

WBALANCE = (BALANCE*2.15)/3
IF WBALANCE GE 100 GOTO GOOD ENDIF

T1 'CONTROL LISTING FOR ABC AND '
'CO'S ACCOUNTS RECEIVABLE'

TABLE1 36 VALUE 'ABCDEFGHIJKLMNQRSTUWXYZ'
'0123456789'
```

Figure 8 Example of Continuation Rules

Naming Conventions

Data names (field names) can be from 2 to 50 characters long, depending on installation options, and must start with an alphabetic character. Data names cannot be the same as a reserved word. For more information, see [Appendix A: Reserved Words](#).

Note: The prefix DYL is reserved for VISION:Results use only.

The data name can be alphanumeric, but it must not contain any punctuation or arithmetic symbols such as:

```
- + * / , ; : . = @ ( )

MM DD YY      DATE
TRANSDATE     DATES
ACCT CODE     AMT FLD
DATEMSTR      FIELDAMT
X1            Q1
```

Figure 9 Example of Data Names (Field Names)

File names can be from one to eight (seven for VSE) characters long and must start with an alphabetic character. The file name can be alphanumeric, but it must not contain punctuation or arithmetic symbols such as:

- + * / , ; : . = @ ()

```
PAYROLL      ARFILE
PAYMSTR      APFILE
```

Figure 10 Example of File Names

Tag names can be from 1 to 10 characters in length and start with an alphabetic character or a number. Tag names cannot be the same as reserved words (see [Appendix A: Reserved Words](#)). The tag name can be alphanumeric and must not contain any punctuation or arithmetic symbols such as:

- + * / , ; : . = @ ()

```
GOTO      LISTDET
GOTO      WRT_RECORD
GOTO      PROCESS_ID
GOTO      FLD_INDIC
GOTO      FLAGS
GOTO      PROCEDURE
GOTO      600
```

```
LISTDET:
WRT_RECORD:
PROCESS_ID:
FLD_INDIC:
FLAGS:
PROCEDURE:
```

Figure 11 Example of Tagnames

The exception is the colon (:), when the tag name is shown as the reference point. The length can be from 2 to 10 characters plus a colon.

Number of Characters per Input Line

The free-form input to VISION:Results is on 80-byte (position) records, the first 72 bytes (VISION:Results default) of which contain the free-form statements. This leaves bytes (positions) 73 to 80 for an identification or sequence field. The default can be changed by the COLUMNS keyword in the OPTION statement. To change the start and end (the minimum is 30 positions) of the VISION:Results free-form, in the free-form input record, use:

```
OPTION COLUMNS 10 60
```

Figure 12 COLUMNS Keyword in the OPTION Statement Example

The example states that all valid entries begin in byte (position) 10 and end in byte (position) 60 of the free-form input record. All other information is considered as extraneous and is not validated. COLUMNS nn nn must be entered on a statement by itself and must be within the bounds of both the old and new free-form columns. However, a statement used for comments and denoted as such with the use of an asterisk (*) must have the asterisk coded in position 1, regardless of the COLUMNS option chosen.

Rules for Comparing Fields

Use the following rules for comparing fields.

Character/Character

Both fields must be less than 256 characters. When the two fields are not the same length, the shorter field is padded with trailing blanks prior to the compare. A logical compare is done on the fields.

Character/Numeric or Packed

The character field's length must be 16 bytes or less. If the character field is longer than 16 bytes, a validation error results. An algebraic compare is done; if the character field contains non-numeric data, a program check occurs.

Character/Binary

Both fields must be the same length. A logical compare is done.

Numeric/Numeric

An algebraic compare is done.

Numeric/Binary

An algebraic compare is done.

Numeric/Packed

An algebraic compare is done.

Packed/Packed

An algebraic compare is done.

Packed/Binary

An algebraic compare is done.

Binary/Binary

Both fields must be the same length. An algebraic compare is done.

Rules for Moving Fields

The following are the rules for moving fields.

Character/Character

A 1- to 32767-byte character field can be moved to a 1- to 32767-byte character field. If you move a shorter field to a longer field, the shorter field is left-justified and padded with blanks. If you move a longer field to a shorter field, the longer field is left-justified and truncation occurs.

When padding occurs, the length of the receiving field is used to determine how many blanks are moved. No checking is done to ensure that an indexed MOVE does not exceed the boundaries of the receiving field. Use the LENGTH keyword to limit the amount of padding.

Character/Numeric

Treated as numeric to numeric.

Character/Packed

Treated as numeric to packed.

Character/Binary

Treated as numeric to binary.

Numeric/Character

A numeric field can be moved to a 1- to 16-byte character field. If the sign of the numeric field is C, the sign is changed to an F in the receiving field. If you move a shorter numeric field to the character field, it is right-aligned in the receiving field and high order zero filling occurs. If the receiving field is shorter, high order truncation occurs. If the numeric field is defined with decimals, they are omitted.

Numeric/Numeric

A numeric field can be moved to a numeric field. Decimal alignment occurs automatically. The sign of the receiving field is the same as that of the sending field. If the receiving field is not large enough to hold the sending field, high or low order truncation occurs.

Numeric/Packed

A numeric field can be moved to a packed field. Data conversion and decimal alignment occur automatically. If the packed field is not large enough to hold the value of the numeric field, high or low order truncation occurs.

Numeric/Binary

A numeric field can be moved to a binary field. Data conversion and decimal alignment occur automatically. If the sending field is unsigned, the receiving field is signed positive. If the binary field is not large enough to hold the value of the numeric field, high or low order truncation occurs and the sign can change.

Packed/Character

A packed field can be moved to a 1- to 16-byte character field. Data conversion occurs automatically. If the sign of the packed field is C, the sign is changed to F in the receiving field. If the character field is not large enough to hold the value of the packed field, high order truncation occurs. If the packed field is defined with decimals, they are truncated. If the character field is larger, high order zero filling occurs.

Packed/Numeric

A packed field can be moved to a numeric field. Data conversion and decimal alignment occur automatically. If the sending field is unsigned, the receiving field is signed positive (C). If the numeric field is not large enough to hold the value of the packed field, high or low order truncation occurs.

Packed/Packed

A packed field can be moved to a packed field. Decimal alignment occurs automatically. If the sending field is unsigned, the receiving field is signed positive (C). If the receiving field is not large enough to hold the value of the sending field, high or low order truncation occurs.

Packed/Binary

A packed field can be moved to a binary field. Data conversion and decimal alignment occur automatically. If the binary field is not large enough to hold the value of the packed field, high or low order truncation occurs and the sign can change.

Binary/Character

A binary field can be moved to a 1- to 16-byte character field. Data is converted to external decimal automatically. If the value is positive, the sign of the character field is set to F to ensure a printable field. If the character field is not large enough to hold the value of the binary field, high order truncation occurs. If the character field is larger, high order zero filling occurs. If the binary field is defined with decimals, they are omitted.

Binary/Numeric

A binary field can be moved to a numeric field. Data conversion and decimal alignment occur automatically. If the sending field is positive, the receiving field is signed positive (C). If the numeric field is not large enough to hold the value of the binary field, high or low order truncation occurs.

Binary/Packed

A binary field can be moved to a packed field. Data conversion and decimal alignment occur automatically. If the sending field is positive, the receiving field is signed positive (C). If the packed field is not large enough to hold the value of the binary field, high or low order truncation occurs.

Binary/Binary

A binary field can be moved to a binary field. Decimal alignment occurs automatically. If the binary receiving field is not large enough to hold the value of the sending field, high or low order truncation occurs and sign changes can take place.

Literals

Literals are occasionally needed in your program. For example:

To move a value to a field:

```
MOVE 'CA' TO STATE  
MOVE 100 TO AMOUNT
```

Figure 13 Move a Value to a Field

To compare a field to a value in a conditional statement:

```
IF AMOUNT GE 500.75 GOTO PROC3 ...
IF STATE EQ 'CA' GOTO PROC4 ...
```

Figure 14 Compare a Field to a Value

To assign an initial value to a field during its definition:

```
FILE FILEOUT OUTPUT FROM FILEOUT
  FIELDA 5 30 PD 2 VALUE 0
  FIELD C 15 50 VALUE 'LOS ANGELES'
WORKAREA
  BASEAMOUNT 5 PD 2 VALUE 250.50
```

Figure 15 Assign an Initial Value

To print a constant on a line of a report:

```
ON CHANGE IN COSTCTR
  LIST '*COST CENTER TOTAL' AT NAME ...
```

Figure 16 Print a Constant on a Line

To print a constant during file print:

```
PRINT 'ERROR 201'
```

Figure 17 Print a Constant

To use a numerical value in an arithmetic expression:

```
AMOUNT=AMOUNT*1.20
COUNT=COUNT+1
NEGVALUE=AMOUNT*-1
```

Figure 18 Use a Numerical Value

Character or Alphanumeric Literals

A character or alphanumeric literal such as 'JONES' or '201-22' is expressed by enclosing it in single or double quotation marks so it is distinguished from a data name.

```
MOVE 'JONES' TO NAME ...
IF STATE EQ 'CA' MOVE 'CALIFORNIA' TO STATENAME ...
PRINT "ERROR 555" ...
```

Figure 19 Character or Alphanumeric Literal

In a literal containing a quotation mark, you must repeat the quotation mark to have the literal accepted.

```
MOVE 'IT'S' TO FIELD A
```

Figure 20 Repeat Quotation Mark

FIELD A now contains IT'S.

Moving Character Literals

To move a character literal to a field, the literal's maximum length is 70. You must code it on one line; it cannot be continued. You can move a 70-byte or smaller literal to a larger field up to a maximum of 32767 bytes. The receiving field is padded with blanks. The receiving field must have a character data type.

```
MOVE 'JONES' TO NAME
```

Figure 21 Move a Character Literal to a Field

If NAME is a 10-byte field, the receiving field contains JONES**BBBBBB**.

Comparing Character Literals

To compare a field to a literal, the literal's maximum length is 20. The field being compared to can have a maximum size of 256 bytes. It must have a character data type. The literal must always be coded to the right of the relational operator (for example, EQ or GT).

To compare a larger-sized field to a literal, before the compare is run, the literal is padded with blanks to make it the same size as the field.

```
IF NAME EQ 'JONES' ...
```

Figure 22 Literal is Padded With Blanks

If NAME is a 10-byte field, then the compare works like this:

```
IF NAME EQ 'JONES      ' ...
```

Figure 23 Results of Literal Padding in [Figure 22](#)

Assigning Character Values to Fields

To define a character type field in an output FILE or WORKAREA statement, assign an initial value to that field using a character literal. If the literal is smaller than the field it is initializing, the literal is left-aligned and padded with blanks.

```
WORKAREA
  FIELD1 10      VALUE 'CODE G'
```

Figure 24 Define a Character Type Field

Because the literal 'CODE G' is only 6 bytes and FIELD1 is 10 bytes in size, four blanks are appended to the value.

The value can be up to 255 bytes in length. You can continue this type of literal onto several lines by breaking it into multiple literals surrounded by quotation marks.

```
WORKAREA
  ALPHANUM 36 VALUE 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
  '0123456789'
```

Figure 25 Breaking it Into Multiple Literals

Here, a 36-byte literal used to initialize ALPHANUM is being continued on a second line.

Listing Literals in a Report

You can print a constant value on a line in report printing. This is the case when you want to identify subtotal or control break lines in the report and want to print, for example, 'DEPARTMENT TOTAL' or 'GRAND TOTAL.' You can code a character type literal in your LIST statement in place of a field name. The actual value prints on the report rather than the value of a named field.

```
ON CHANGE IN DEPARTMENT
  LIST 'DEPARTMENT TOTAL' AT NAME SUM SALARY
```

Figure 26 Code a Character Type Literal in Your LIST Statement

Here, when a control break occurs caused by a change in the value of DEPARTMENT, a line is printed containing the constant 'DEPARTMENT TOTAL' along with the other fields specified.

The maximum size of a literal appearing in a LIST statement is 30. It must be coded on one line and cannot be continued.

Printing a Literal in a FILE PRINT

The file print feature (PRINT, HEXPRINT, HEX, and so on) can be used to print records or portions of records. You can print a message or identifier to the SYSPRINT (z/OS) or SYSLSST (VSE) file.

```
PRINT 'ERROR 321'
PRINT 'BEGIN FILEA PRINT'
```

Figure 27 Printing a Literal in a FILE PRINT

Your message literal can be up to 20 bytes long.

Numeric Literals

A numeric literal such as 0, 595, 300.50, or -1 can be expressed without quotation marks. It can include a leading minus sign (if negative), commas, and a decimal point. It can contain a maximum of 16 digits. The total length of the numeric literal must not exceed 20 bytes. It can be used in operations with numeric, packed, or binary type fields.

```
MOVE 0 to AMOUNT
IF AMOUNT GE 500.50 GOTO ...
COUNT=COUNT+1
CALC=FIELDA*37.5/100
```

Figure 28 Numeric Literals

Decimal alignment is automatic.

```
MOVE 400 TO AMOUNT
```

Figure 29 Example of Decimal Alignment

If AMOUNT is a 2-decimal position field, the 400 is actually moved as though it were 400.00.

```
WORKAREA
  FIELDA 5 2 PD VALUE 0
```

Figure 30 2-Decimal Position Field FIELDA

The 2-decimal position field FIELDA is completely zeroed.

Moving Numeric Literals

You can move a numeric literal to a numeric, packed, or binary field.

```
MOVE 9,999.99 TO AMOUNT
MOVE -700 TO AMOUNT2
```

Figure 31 Move a Numeric Literal

AMOUNT or AMOUNT2 could have been defined as a numeric (NU), packed (PD), or binary (BI) field.

Comparing Numeric Literals

You can compare a numeric, packed, or binary field to a numeric literal. The literal must appear to the right of the relational operator (for example, EQ or GE).

```
IF AMOUNT GE 0 ...
IF AMOUNT2 GT 9,999.99 ...
```

Figure 32 Compare a Numeric Literal

Using Numeric Literals in Arithmetic Expressions

Numeric literals can be used in arithmetic expressions with other numeric literals, numeric packed, or binary fields.

```
AMOUNT=AMOUNT*1.10  
COUNT1=COUNT1+1  
COUNT2=(AMOUNT-1.75)/4  
NEGAMT=AMOUNT*-1  
AMOUNT2=FIELDA+1,245,373.25
```

Figure 33 Numeric Literals in Arithmetic Expressions

Assigning Values to Numeric, Packed, or Binary Fields

To define a numeric, packed, or binary field in an output FILE or WORKAREA statement, assign a value to that field using a numeric literal:

```
WORKAREA  
  FIELDA 5 PD 2    VALUE 0  
  FIELDB 3 NU 1    VALUE 3.1  
  FIELD C 1 BI     VALUE 127  
  FIELD D 1 PD     VALUE -1
```

Figure 34 Assigning Values to Numeric Fields

You can assign a value up to nine decimal positions:

```
FIELDE 6 PD 9    VALUE 1.123456789
```

Figure 35 Example of Assigned Values

Hexadecimal Literals

You can express a value that cannot be represented easily using alphanumeric or special character notation. Express these values using hexadecimal notation. It consists of representing each byte by a pair of characters in which the value can be in the range of 0-9, A-F. Precede the start of such a literal by X' and end it with a '. Using this notation, the character 'A' is expressed as X'C1.' The literal must be coded on one line; it cannot be continued to another line.

You cannot use this type of literal in arithmetic operations.

Moving Hexadecimal Literals

You can move up to a 9-byte hexadecimal literal to a field. The receiving field must have a character or binary data type and must be the same size as represented by the literal.

```
MOVE X'0102' TO FIELDG
```

Figure 36 Hexadecimal Literal Move

FIELDG must be a 2-byte character field.

```
MOVE X'FF' TO SWITCH
```

Figure 37 Hexadecimal Literal Move 2-byte character field

SWITCH must be a 1-byte character field.

Comparing Hexadecimal Literals

You can compare up to a 9-byte character field to a hexadecimal literal. The field must be the same size as represented by the literal. The literal must be coded to the right of the relational operator (for example, EQ or NE).

```
IF FIELD1 EQ X'FFFFFF' GOTO ...
```

Figure 38 Hexadecimal Literal Compare

FIELD1 must be a 3-byte character field.

Assigning Values to Fields Using Hexadecimal Notation

To define a character type field in an output FILE or WORKAREA statement, assign an initial value to that field using a hexadecimal literal. The initialized field can be up to 9 bytes in size. The field must be the same size as represented by the literal.

```
WORKAREA  
TABLE1 9 CH VALUE X'010203040506070809'  
SWITCH 1 CH VALUE X'FF'
```

Figure 39 Initial Value Field Using a Hexadecimal Literal

Chapter 4: Using the OPTION Statement

The OPTION statement provides you with a great amount of flexibility in defining the application environment and parameter settings. You can use the OPTION statement to supply information about the installation's operating environment, specify your user preferences, or override the VISION:Results site defaults as specified in the DYLINSTL. The site defaults for all parameters are shown in the Default Options Page display. For information on the DYLINSTL macro, see the *Advantage VISION:Results for z/OS Installation Guide*.

Note: Keywords in **bold** apply only to VSE environments.

You can use OPTION keywords to:

- Customize program input and processing

CDLOAD	NOVSAMCAT	TSIZE
COLUMNS	NOVSEALL	USERDEFAULT
CONVENTIONAL	NOVSEDISK	!USERINXnn
DATA	NOVSETAPE	VSAMCAT
DISK	NUMCHAR	VSEALL
\$\$KEYLEVEL	NUMPD	VSEDISK
NOCDLOAD	QLF	VSETAPE
NONUMCHAR	QUALIFIERS	ZDIVAB
NONUMPD	STRUCTURED	ZDIVORG
NOQLF	STRUCTURED2	ZDIVRC
NOQUALIFIERS	SYSnnn	

- Customize program output and reports

COBAPOS	NOCOBXSIGN	PDSREPN
COBXSIGN	NODYL4YEAR	PRINT
CURRENCY	NOPRINTCTRS	PRINTCTRS
DYL4YEAR	NOPRINTDIGITS	PRINTDIGITS
HEX	NOSYSBLOCK	REPORTFILE
HEXPRINT	NULLOFF	SYSBLOCK
LCPRINT	NULLON	TIMECOLON
NOCBAPOS	PDSREPA	TIMEDOT

- Provide environment specifications

CATPLANID	LE	NDVRTYP
CATSYSID	NDVRCOM	NOCOB2NR
COB2NR	NDVRENV	NOLE
DB2PLANID	NDVRSTG	PLANID
DB2SYSID	NDVRSUB	STATSQL
DYNAMDB2	NDVRSYS	SYSTEMID

- Optimize resources

FREEZE	FREEZDD	RESTORE

- Provide access to companion products

260	IQBATCH	NOSPOOL
EXCEL	NOEXCEL	QUIKJOB
EXCELPAT	NOEXCELPAT	SPOOL

- Program listing and formatting options

LONG	NOTOTAL	TOTAL
NOOPTLIST	OPTLIST	

- Aid application debugging and documentation

DMAP	NOPRINTEP	PRINTERR
\$\$DUMP	NOSORTAB	PRINTERERROR
EXPRTErr	NOXREF	VERIFY
MSGCSECT	\$\$PCB	XREF
NOEXPRTErr	PRINTEP	XREFA

OPTION statements are not required, but if present must precede all other statements in a program except blank lines, comments, COPY, REPORT, STATEON, and STATEOFF commands. Exception: The OPTION COLUMNS statement can be anywhere in your program.

You can specify multiple keywords on an `OPTION` statement or use multiple `OPTION` statements. The `OPTION` statement syntax on the following pages is in Backus-Naur-Form (BNF) notation:

- A vertical bar `|` separates mutually exclusive alternatives.
- Braces `{ }` are used to group related items that are alternatives. One item must be chosen.
- Brackets `[ ]` are used to indicate optional items.
- Uppercase and special characters must be specified exactly as shown.
- Lowercase characters require a specific value to be substituted in their place.
- Parentheses must be typed exactly as shown.
- Keywords and operands between a set of `!` and `!...` indicate that the repetition of what is between the `!` and `!...` is allowed.
- Bold text inside program examples indicates VSE only.

Program Input and Processing Options

This section describes the program input and processing options available using the `OPTION` statement.

CDLOAD Facility

Note: This section applies to VSE only.

```
OPTION [CDLOAD | NOCDLOAD]
```

The `CDLOAD` option loads the named subroutine using the VSE `CDLOAD` facility, rather than `LOAD`, for user programs referenced in a `CALL` statement.

The subroutine is loaded into the partition's dynamic area rather than into storage reserved by the compiler. Because of this, the size parameter in the `CALL` statement is not needed. VSE determines the storage required. If the size parameter previously used reserved additional storage at the end of the phase for the subroutine's use, this storage is no longer available. In this case, the `CALL` statement must specify `NOCDLOAD` and use the size value as before. `CDLOAD` or `NOCDLOAD` in the `OPTION` statement overrides the `DYLINKSTL` specification.

For information on calling the `CDLOAD` facility, see [Chapter 13: Using the CALL Command](#).

Input Columns

```
OPTION COLUMNS start end
```

The program input to VISION:Results is an 80-byte record. The default configuration uses the first 72 bytes for the VISION:Results free-form statements, leaving bytes 73 to 80 for a programmer identification or sequence field. This free-form statement input location can be customized using the COLUMNS keyword.

Enter the "OPTION COLUMNS start end" statement by itself, keeping it within the bounds of both the default and newly specified column boundaries. A new free-form header is printed for the statements that follow in the program listing. The value specified for 'end' may include columns through 80. The minimum number of usable columns is 30.

```
OPTION COLUMNS 10 60
```

Figure 40 Example: COLUMNS Start End Statement

Data entered prior to position 10 and after position 60 in the input record is ignored. There are 51 usable columns.

Programming Mode

```
OPTION [ CONVENTIONAL | STRUCTURED | STRUCTURED2 | USERDEFAULT ]
```

VISION:Results provides various programming modes you can use depending upon your requirements. Once the mode has been selected, it remains in effect for all report requests in a multiple report run unless it is changed.

CONVENTIONAL is the default mode and sets the mode to standard VISION:Results.

STRUCTURED mode allows the use of structured features in VISION:Results such as PERFORM. To adhere to strict structured constructs, this mode disables the usage of the GOTO, ACCEPT, and REJECT commands. For a complete description of this feature, see the *Advantage VISION:Results for z/OS Getting Started*.

STRUCTURED2 mode allows all structured features, as well as the GOTO, ACCEPT, and REJECT commands, establishing a less restrictive environment than OPTION STRUCTURED.

USERDEFAULT mode assumes certain defaults to simplify programming.

For information, see the *Advantage VISION:Results for z/OS Getting Started*.

Input Literal Data

```
OPTION DATA 'literal'
```

The DATA keyword can be used to supply a literal as input data to a VISION:Results program from within the program source. It is placed into the reserved word DYLPARM. z/OS users can use the PARM facility of the EXEC statement. For a definition of DYLPARM, see the *Advantage VISION:Results for z/OS Getting Started*. The literal is specified between quotation marks ('), can be numeric or alphanumeric, and cannot be longer than 38 characters.

```
OPTION DATA '01/15/01'  
FILE ARMSTR FB 100 1000  
  ACCTNO 5 BALANCE 5 170 PD 2 A  
WORKAREA  
  TODAY 8  
MOVE DYLPARM TO TODAY  
LIST ACCTNO BALANCE TODAY
```

Figure 41 Example: Passing a Variable Using the DATA keyword

Old Release Keyword Equivalents

```
OPTION $$KEYLEVEL { 2.2 | 2.5 | 2.8 | 3.0 | 3.5 | 4.0 | 5.0 }
```

The \$\$KEYLEVEL option backdates the release of VISION:Results so that new reserved words are ignored (keyword table entries). This allows older programs created under a previous release of VISION:Results to run even if some of the data names conflict with reserved words in the newer releases.

Modifying IF NUMERIC Test

```
OPTION [NUMCHAR|NONUMCHAR]
```

NUMCHAR restricts the IF NUMERIC test to check for only F0 through F9 in all bytes, disallowing a sign in the low order digit for NU or CH fields. For PD fields, NUMCHAR checks the sign field for x'C'.

OPTION NONUMCHAR reverts to the standard test.

These keywords override the DYLINSTL macro parameter setting.

IF NUMERIC Test for PD Fields

```
OPTION [NUMPD|NONUMPD]
```

NUMPD allows the IF NUMERIC statement to be used for PD fields. NONUMPD causes the DYI-480E message to be issued if a PD field is used with an IF NUMERIC statement.

These keywords override the DYLINSTL macro parameter setting.

Qualification of Data Names

```
OPTION [QLF | QUALIFIERS | NOQLF | NOQUALIFIERS]
```

VISION:Results provides a qualification of data name feature that allows you to use non-unique data names in a program. A qualifier can be a file name, a table name, a named WORKAREA, or a used IDname. For more information, see [Chapter 18: Multiple Reports and Multiple Requests](#).

NOQLF or NOQUALIFIERS may be specified to negate the qualification option.

These keywords override the DYLINSTL macro parameter setting.

Allocating Memory for Tables and Arrays

```
OPTION TSIZE { nnM | nnnnnK }
```

If you define a table or an array, VISION:Results automatically acquires additional memory to store the table or array. The TSIZE parameter in the OPTION statement allows you to control the amount of storage reserved.

For example, if you need 2 KB of memory for the entries in one table and 4 KB for another, to allocate enough space for both tables, code:

```
OPTION TSIZE 6K
```

The minimum TSIZE allowed is 2 KB; the maximum is 63000 KB. If the TSIZE is not specified, a default of 20 KB is used.

You can use the TABLE/ARRAY processor of VISION:Results to create and access tables residing above the 16 MB line. Use this option to table and search large amounts of data.

Here are some guidelines for deciding how much memory to allow:

- Round up the number of bytes of data you expect to store in a given table to the next multiple of 1024 (1 KB).
- Be generous in your estimate unless you know exactly how much data must be tabled. Remember that tables can grow as your customer base grows.
- Variable-length tables require an additional 2 bytes per entry to hold the entry length.
- Arrays require the product of the element length and all the dimensions.
- If insufficient storage is available to satisfy the memory required, an 80A system abend or a 602 completion code (VSE) results. If this happens, increase your region/partition size or decrease your TSIZE.

VSE Disk Work File

Note: This section applies to VSE only.

```
OPTION SYSnnn [filename]
```

```
OPTION DISK {2311 | 2314 | 3330 | 3340 | 3350 | 3375 | 3380 | FBA }
```

VISION:Results uses a disk work file. For VSE users only, it is necessary to specify the type of disk device being assigned to the work file if the default cannot be used.

If a disk type is not specified, VISION:Results defaults to 3330. The default can be changed during installation of VISION:Results. FBA represents fixed block architecture devices.

For VSE users on VSE systems prior to r2.1 only, the default symbolic unit and file name of this work file may be changed. The default symbolic unit is SYS004. The default file name is IJSYS04. To change the symbolic unit, use the SYSnnn keyword, where nnn is the new symbolic unit. The filename parameter may be used to specify a new file name. The file name can be one to seven alphanumeric characters long, starting with an alphabetic character.

An example of an override default VSE work file system number is:

```
OPTION SYS010
```

An example of an override default VSE work file system number and file name is:

```
OPTION SYS010 DYLOWRK
```

User-Defined Index Fields

```
OPTION !USERINXnn dataname!
```

You can define your own index fields if you want to extend the number of indices beyond the standard INW, INX, INY, and INZ. VISION:Results allocates a 2-byte binary field for each specified name. You must specify the names of these fields on the OPTION statement and they cannot be defined in another VISION:Results statement. Once defined, you can specify the user index in place of INW, INX, INY, or INZ in all statements that currently allow indexing.

A maximum of 99 user index fields can be defined; nn is a number from 01 to 99.

The data name must be 2 to 10 characters in length, starting with an alphabetic character.

VSAM File Characteristics

```
OPTION [VSAMCAT | NOVSAMCAT]
```

When the VSAMCAT keyword is specified in the OPTION statement, file characteristics of VSAM files are obtained from the VSAM catalog entry that was created for the file. You do not need to specify the file type and characteristics of the VSAM file. You must specify the DD statement or DLBL statement, even for a FREEZE or verify run.

These keywords override the DYLINSTL macro parameter setting.

This option must be in effect when processing VSAM RLS files.

Match File Attributes for VSE Sequential Files

Note: This section applies to VSE only.

```
OPTION [VSEALL | NOVSEALL]  
OPTION [VSEDISK | NOVSEDISK]  
OPTION [VSETAPE | NOVSETAPE]
```

VISION:Results constructs DTFs (Define The File) that match your actual FILE statement attributes. In place of the U (undefined) in the RECFM (record format), the DTF contains one of the following appropriate values:

```
{F | FB | V | VB}
```

Note: In previous releases of VISION:Results, the old method called for DTFs with a RECFM of U to be utilized when processing a sequential file under VSE.

OPTION VSETAPE uses the new DTF method for VSE tape files only. Disk files use the default method. Conversely, OPTION VSENOTAPE uses the old method.

OPTION VSEDISK constructs DTFs for VSE disk files only. Tape files use the default method. OPTION NOVSEDISK uses the old method.

Use OPTION VSEALL when you want VISION:Results to construct DTFs for all VSE files, both tape and disk. OPTION NOVSEALL ignores the default method for all VSE files, both tape and disk.

These keywords are not allowed on an OPTION FREEZE statement, and a warning message is issued. In an OPTION RESTORE statement or a straight OPTION statement, these keywords provide the default value for the application being restored.

These keywords override the DYLINSTL macro parameter setting.

Divide by Zero Condition

```
OPTION [ ZDIVRC | ZDIVAB | ZDIVORG ]
```

When the ZDIVRC keyword is specified, if a divide by zero condition occurs, then the 0 is changed to a 1, processing continues, and a return code of 4 is issued.

When the ZDIVAB keyword is specified, if a divide by zero condition occurs, then a system ABEND occurs.

If the DYLINSTL parameter ZDIVAB=Y or ZDIVAB=R has been set as the installation default and the ZDIVORG keyword is specified on the OPTION statement, the system default reverts to its original state. The 0 is changed to a 1 and processing continues.

Program Output Options

This section describes the program output options available using the OPTION statement.

COBAPOS Option

OPTION [COBAPOS | NOCOBAPOS]

The COBAPOS keyword stores a single apostrophe whenever two consecutive apostrophes are coded on a COBOL VALUE clause. NOCOBAPOS stores two apostrophes whenever two consecutive apostrophes are coded on a COBOL VALUE clause.

These keywords override the DYLINSTL macro parameter setting.

COBXSIGN Option

OPTION [COBXSIGN | NOCOBXSIGN]

If the COBXSIGN keyword is used, COBOL data names included with COPY statements or instream definitions with the SIGN IS SEPARATE clause will be stored as character data with the field length increased to include the sign position.

NOCOBXSIGN overrides the DYLINSTL macro if it is set as CBXSIGN=Y.

Currency Symbol Substitution

OPTION CURRENCY {C'x' | X'nn'}

The CURRENCY keyword on the OPTION statement overrides the CURRENCY parameter set in the DYLINSTL macro, which assigns a symbol to be used instead of \$ for the currency symbol. The symbol may be defined as any character using the format C'x' where x is any alphanumeric or special character or X'nn' where nn is any pair of valid hexadecimal characters.

Four-Digit Year Support

OPTION [DYL4YEAR | NODYL4YEAR]

Specifying DYL4YEAR keyword creates a 4-digit year heading in SYSPRINT or SYSLST. When used with the USERDEFAULT option, the DYL4YEAR keyword also produces a 4-digit year in the TITLE line.

The default is a 2-digit year format. This option overrides the DYL4YR parameter in the DYLINSTL macro. For information on the USERDEFAULT mode, see the *Advantage VISION:Results for z/OS Getting Started*.

FILE PRINT Options

```
OPTION [HEXPRINT filename | PRINT filename| HEX filename| LCPRINT filename]
```

These options specify which format is to be used when printing a file. The choices HEXPRINT, PRINT, LCPRINT, or HEX are mutually exclusive. Printing a file is completely independent of report printing.

You can specify any file to be printed. Using this option prints an entire file unless a REJECT command is encountered in the program logic to omit records or there is not an exit from the logic using ACCEPT or an implied ACCEPT. If portions of the file are to be printed, it is better to use the commands HEX, PRINT, and HEXPRINT in the procedure logic.

- The HEXPRINT keyword prints a file in hexadecimal and graphics.
- The PRINT keyword prints a file in graphics only.
- The HEX keyword prints a file in hexadecimal only.
- The LCPRINT keyword prints a file in graphics only, but does not translate normally unprintable characters to blanks. The keyword LCPRINT is necessary when lowercase characters are to be printed.

The OPTION keywords for HEX, PRINT, LCPRINT, and HEXPRINT and the procedure logic HEX, PRINT, LCPRINT, and HEXPRINT commands are mutually exclusive. You cannot specify file print in both the OPTION statement and the procedure logic.

Output files can only be file printed when they are output from themselves. The file name can be one to eight (seven for VSE) characters long. The file must begin with an alphabetic character.

```
OPTION HEXPRINT ARMSTR FILE ARMSTR FB 100 1000
OPTION HEXPRINT ARFILE
OPTION HEX ARFILE
```

```
FILE ARFILE FB 352 5280
ACCOUNT 2 182
IF ACCOUNT EQ 'WO'
PRINT ARFILE ELSE
HEXPRINT ARFILE ENDIF
```

```
FILE ARFILE FB 352 5280
ACCOUNT 2 182
IF ACCOUNT EQ 'WO'
PRINT ARFILE ENDIF
```

or

```
OPTION PRINT ARFILE
FILE ARFILE FB 352 5280
ACCOUNT 2 182
IF ACCOUNT EQ 'WO'
ACCEPT ELSE REJECT ENDIF
```

Figure 42 Examples: Using FILE PRINT Options

Null Output File Control

OPTION [NULLON | NULLOFF]

If an output file is defined but no records are written to it, the NULLON keyword forces VISION:Results to OPEN and CLOSE the file when the program ends. This allows subsequent programs using the file to depend on the existence of the file. If this option is established as the installation default using the DYLINSTL macro parameter, OUTFILE=Y, then keyword NULLOFF can be used to reverse the effect.

PDS Replace

OPTION [PDSREPA | PDSREPN]

When WRITEDIR memname REPLACE is executed in a program and the member is not present, the PDSREPA keyword sets the status byte to A if the member is successfully added. If the PDSREPN keyword is specified, an N is returned in the status byte if the member is successfully added.

These keywords override the DYLINSTL macro parameter setting.

Counter Display

OPTION [PRINTCTRS | NOPRINTCTRS]

PRINTCTRS allows you to print the values and labels for all 126 counters at the end of the execution for each request even if the value is zero. NOPRINTCTRS suppresses the counter display.

Extended Arithmetic Report Field Display

OPTION [PRINTDIGITS | NOPRINTDIGITS]

Extended arithmetic capability allows VISION:Results to handle very large numbers with greater internal precision. You can either list your data names in the extended arithmetic format (21 digits and 9 decimal places) or non-extended format (10 digits and 5 decimal places). This is specified by using keyword PRINTDIGITS (list in extended format) or NOPRINTDIGITS (list in non-extended format). PRINTDIGITS is the default. This option affects the LIST statement only.

These larger summed numeric fields take up more space on a report line than VISION:Results r2.2 or earlier. Previously existing VISION:Results programs that now exceed the maximum print line length must use OPTION NOPRINTDIGITS to cause the report to be composed as in the old releases.

VSE Report File Name

Note: This section applies to VSE only.

```
OPTION REPORTFILE filename
```

To print a VSE report file, use the REPORTFILE keyword with a filename parameter where 'filename' consists of the one- to seven-character name of the file to be printed. The file must have a carriage control character (machine or ASA) in the first byte of each record. This byte is used to control line spacing and page ejecting. The file must be defined by a FILE statement. This option cannot be used with any of the file print options (OPTION HEX, OPTION PRINT, OPTION LCPRINT, or OPTION HEXPRINT).

```
OPTION DISK 3380
OPTION REPORTFILE ARREPT
FILE ARREPT FB 133 1330
```

Figure 43 Example: Print a VSE Report File

System-Determined Block Size Support for z/OS

```
OPTION [SYSBLOCK | NOSYSBLOCK]
```

You can allow the operating system to calculate the appropriate block size of an output file created by VISION:Results when no block size information is provided in the program.

For z/OS to calculate the block size:

- A non-zero block size must not be coded on the DD statement in the JCL associated with the file.
- The block size must not be specified on the FILE statement.
- The BY parameter of the PICNSAVE statement must not be used.

These keywords override the DYLINSTL macro parameter setting.

DYLTIME Format Change

```
OPTION [TIMECOLON | TIMEDOT]
```

The TIMECOLON keyword causes DYLTIME to list in format HH:MM:SS.

The TIMEDOT keyword formats DYLTIME to list as HH.MM.SS.

These keywords override the DYLINSTL macro parameter setting.

Environment Specification Options

This section describes the environment specification options available using the `OPTION` statement.

COBOL II Environment

```
OPTION [COB2NR | NOCOB2NR]
```

Specifying `NOCOB2NR` overrides the `DYLINSTL` parameter setting, `COB2NR=Y`, which allows proper execution when calling COBOL II subroutines compiled with the `NORES` option. Conversely, indicating `COB2NR` reinstates this feature.

IBM Language Environment (LE) Support

```
OPTION [LE | NOLE]
```

The `LE` keyword can be used to override the `DYLINSTL LE` setting, which initializes the IBM Language Environment (LE). This allows your program to `CALL` subroutines written in any LE-compliant language, including COBOL, C, PL/I, FORTRAN, and Assembler as well as the IBM LE Service Routines such as `CEEDATE` and `CEEDAYS`.

If `LE` is specified and `DYLINSTL` parameter `COBENV=Y`, a warning message will be issued, stating the `COBENV` option will be ignored.

`LE` can be specified with `OPTION FREEZE` and `OPTION RESTORE` to override `DYLINSTL LE=N`. Note that the `RESTORE` with `LE` works only if `LE` was enabled when the program was frozen either via `DYLINSTL LE=Y` or `OPTION LE`. If `LE` was not enabled on the `FREEZE`, then a `S0C7` abend can occur if `LE` is encountered on the `RESTORE`.

AllFusion® Endeavor® Environment Variables

<code>OPTION NDVRENV</code>	<code>xxxxxxxx</code>	Endeavor environment name.
<code>OPTION NDVRSYS</code>	<code>xxxxxxxx</code>	Endeavor system name.
<code>OPTION NDVRSUB</code>	<code>xxxxxxxx</code>	Endeavor subsystem name.
<code>OPTION NDVRTYP</code>	<code>xxxxxxxx</code>	Endeavor subsystem type.
<code>OPTION NDVRSTG</code>	<code>n</code>	Endeavor environment stage ID.
<code>OPTION NDVRCOM</code>		Print copied Endeavor element attributes.
<code>OPTION NONDVRCOM</code>		Suppress copied Endeavor element attributes

Use `OPTION` keywords to specify values for the Endeavor environment variables.

If no environment name is provided using the OPTION or DYLINSTL parameter, an error will be issued. If the remaining Endeavor attributes are omitted from the OPTION statement, the element selected will be the first occurrence of the element for any system/subsystem/type and stage for the specified environment. If DYLINSTL parameter NDVRCOM=Y is specified, or if OPTION NDVRCOM is specified, information comments will be printed indicating the attributes of the element retrieved with the copied records. OPTION NONDVRCOM will override DYLINSTL parameter NDVRCOM=Y and suppress these comments.

Options for Advantage™ VISION:Results® Interface™ to DB2

The following options are available to licensed users of VISION:Results Interface to DB2.

Use DB2 Table Definitions

```
OPTION [CATPLANID planname] [CATSYSID ssid]
```

Use CATPLANID and CATSYSID keywords to specify PLAN and subsystem ID when using the COPYDB2 command to copy DB2 table definitions for use as VISION:Results data names.

To change the OPTION statement overrides, you must use the JCL overrides described in [COPYDB2 Requirements on page 346](#)

For VISION:Results Interface to DB2 (Dynamic)

```
OPTION DYNAMDB2    [DB2PLANID {dataname | 'literal'}]  
                  [DB2SYSID {dataname | 'literal'}]
```

Use DYNAMDB2 to override the default DB2 plan and subsystem ID in dynamic runs. For information about OPTION DYNAMDB2, see the *Advantage VISION:Results Interface to DB2 Reference Guide*.

For VISION:Results Interface to DB2 (Static)

```
OPTION STATSOL modulename [PLANID planname SYSTEMID ssid]
```

Use STATSOL to override the default DB2 plan and subsystem ID and specify the module name of a frozen program to be created. For information about OPTION STATSOL, see the *Advantage VISION:Results Interface to DB2 Reference Guide*.

Resource Optimization Options

This section describes the resource optimization options available using the OPTION statement.

Optimization

```

OPTION FREEZE modulename [DATA 'literal' ]
                        [DISK type ]
                        [SYSnnn filename ]
                        [NOTOTAL ]
                        [PRINTERROR | PRINTERR]
                        [RESTART nn ]

OPTION FREEZDD ddname

OPTION RESTORE modulename [nn] [[RESTART nnnn] [DELIM [C'c' | X'xx']]

```

Frequently run programs can be frozen and restored so that the program does not have to be validated and compiled each time it is executed using the FREEZE and RESTORE options.

FREEZE translates a set of statements into object code and places the module in a library for later execution. The module named after the FREEZE keyword must then be link edited to a load library (z/OS) or a core image library (VSE) in the same manner as output produced by compilers and assemblers.

With OPTION FREEZE you can specify a 1- to 8- character ddname for SYS280FZ by specifying it with the FREEZDD keyword. This may be on its own OPTION statement, or included as an additional parameter on the OPTION FREEZE statement.

For example, to generate the frozen module MYPROG and write it to the library with the ddname of FREEZLIB, you can code:

```
OPTION FREEZE MYPROG FREEZDD FREEZLIB
```

RESTORE is used to retrieve the load module from storage and execute it.

You can replace the character delimiter that is specified on the Report statement. This affects only programs that were frozen with the DELIM parameter on the Report statement. You can specify DELIM only on the OPTION statement with RESTORE.

The only keywords allowed to be used with RESTORE are:

COB2NR	NOLE	RESTART nn
DATA literal	NOOPLIST	SYSnnn filename
DELIM [C'c' X'xx']	NOTOTAL	VSEALL
DISK type	NOVSEALL	VSEDISK
DYL4YEAR	NOVSEDISK	VSETAPE
LE	NOVSETAPE	\$\$DUMP
NOCOB2NR	OPTLIST	\$\$PCB
NODYL4YEAR	PRINTERROR	

For specific information about the FREEZE and RESTORE features, see the *Advantage VISION:Results for z/OS Toolkit Reference Guide*.

```
OPTION FREEZE WKMSTR
FILE ARMSTR FB 100 1000
  ACCTNO 5 BALANCE 5 170 PD 2 A
WORKAREA
  PARMINFO 60
  TODAY 8 1
MOVE DYLPARM TO PARMINFO
LIST ACCTNO BALANCE TODAY
```

Figure 44 Example: Freeze a Program

```
OPTION RESTORE WKMSTR DATA '01/15/01'
```

Figure 45 Example: Restore a Frozen Module

Companion Product Options

This section describes the companion product options available using the OPTION statement.

VISION:Sixty

Note: For licensed VISION:Sixty users only.

```
OPTION 260
```

The 260 option allows a VISION:Sixty source program to be executed under VISION:Results. When 260 is specified as a keyword, no other keyword is allowed. The OPTION 260 statement must be the first statement in the program. The VISION:Sixty program source is placed after this statement.

OPTION 260 is non-functional in systems where VISION:Sixty has not been licensed. It is valid only for licensed VISION:Sixty users.

```
OPTION 260
:
:
your VISION:Sixty program
:
:
```

Figure 46 Example: Submit VISION:Sixty Program Through VISION:Results

VISION:Excel Keywords

Note: For licensed VISION:Excel users only.

```
OPTION [EXCEL|NOEXCEL]
```

The EXCEL keyword activates all the VISION:Excel commands and keywords. Programs with data names that are the same as VISION:Excel commands and keywords will be interpreted as VISION:Excel commands and keywords.

The NOEXCEL statement deactivates all the VISION:Excel keywords. This allows programs with the same data names as VISION:Excel to execute without conflict.

The site default is set in the DYLINSTL macro.

This OPTION is non-functional in systems where VISION:Excel has not been licensed.

VISION:Excel Display

Note: For licensed VISION:Excel users only.

```
OPTION [EXCELPAT | NOEXCELPAT]
```

EXCELPAT allows you to print the VISION:Excel patch report for each request in VISION:Excel applications in the AUDPRINT output dataset. NOEXCELPAT suppresses the display. VISION:Excel patches will be displayed in the AUDPRINT dataset.

This OPTION is non-functional in systems where VISION:Excel has not been licensed.

VSE Spooling VISION:Excel Output

Note: For licensed VISION:Excel users only.

Note: This section applies to VSE only.

```
OPTION [SPOOL | NOSPOOL]
```

The SPOOL option is useful when using VISION:Excel to request multiple letters or to print VISION:Excel output at a more convenient time.

For example, when generating two different letters, each printed on a different letterhead, they could be spooled and selected later for printing on the correct stationery.

```
OPTION SPOOL DATA '01/15/99'
```

Figure 47 Example: SPOOL Option

This OPTION is non-functional in systems where VISION:Excel has not been licensed.

VISION:Inquiry Bridge

Note: For licensed VISION:Inquiry users only.

```
OPTION IQBATCH name
```

IQBATCH in the OPTION statement allows a VISION:Results request to provide the name of the load module to be used in implementing the VISION:Inquiry Batch-Results bridge.

This OPTION is non-functional in systems where VISION:Inquiry has not been licensed.

VISION:Report

Note: For licensed VISION:Report users only.

```
OPTION QUIKJOB
```

The QUIKJOB option allows a VISION:Report source program to be executed under VISION:Results. When QUIKJOB is specified as a keyword, no other keyword is allowed. The OPTION QUIKJOB statement must be the first statement in the program. The VISION:Report program is placed after this statement.

OPTION QUIKJOB is non-functional in systems where VISION:Report has not been licensed.

```
OPTION QUIKJOB
:
:
your VISION:Report program
:
:
```

Figure 48 Example: Submit a VISION:Report Program Through
VISION:Results

Program Listing and Formatting Options

This section describes the program listing and formatting options available using the OPTION statement.

Number of Lines per Page

```
OPTION nn LONG
```

VISION:Results defaults to 55 lines per page for both the VISION:Results source listing and any file prints (HEXPRINT, HEX, PRINT, LCPRINT, and REPORTFILE). You can change the number of lines per printed page using the LONG keyword, where 'nn' is number of lines per page. The minimum 'nn' value allowed is 13. This keyword has no effect on the number of lines per page for any report being printed.

The REPORT statement must be used to change the number of lines per page for a report.

The example shown in [Figure 49](#) changes the number of lines printed per listing page to 60.

```
OPTION 60 LONG HEXPRINT ARFILE
FILE ARFILE FB 352 5280
ACCOUNT 2 182
IF ACCOUNT EQ 'IO' ACCEPT
ELSE REJECT ENDIF
```

Figure 49 Example: Change Number of Lines per Page

Default Options Page Display

```
OPTION [OPTLIST | NOOPTLIST]
```

The DYLINSTL macro parameter OPTLIST determines whether the Default Options page is included in the listing at the end of the program. To override the site default, and either print or suppress this display, use the OPTLIST or NOOPTLIST keywords.

Automatic Control Totals

```
OPTION [TOTAL | NOTOTAL]
```

At the end of a run, VISION:Results automatically prints various counts such as DYLCOUNTnn, record and character counts, block counts, fixed blank counts, and divide by zero occurrences. If the DYLINSTL macro parameter NOTOTAL=Y or the NOTOTAL keyword is specified on the OPTION statement, all control totals are suppressed. To override DYLINSTL macro setting of NOTOTAL=Y and print control totals at execution time, use OPTION TOTAL.

Debugging

This section describes the options available for debugging.

Cross-Reference Listings

```
OPTION [ DMAP XREF XREFA ]  
OPTION NOXREF
```

There are multiple options available to assist with program analysis. You can request a cross-reference listing of data names or by area, location, size, and statement number. The cross-reference display is produced at the end of each report request. When a cross-reference option is selected, it remains in effect for all report requests in a multiple report run until it is changed. For more detail, see [Chapter 5: Data Name Qualification](#) and *Advantage VISION:Results for z/OS Messages and Codes*.

Use XREF for cross referencing referenced data names only. The listing is produced at the end of each report request, including the data names from the program in alphabetic sequence as well as the size of the field, the data type, the number of decimals, the VISION:Results statement number where the field was defined, and the VISION:Results statement numbers where the field was referenced.

Use XREFA for cross referencing all data names with the data names in alphabetical order within the corresponding qualifier.

Important! If you have a large number of unreferenced data names, the time required to sort them prior to producing the cross-reference listing can be substantial.

The Data Name Map (DMAP) option prints the data names and their related information in the sequence of Area, Location, Size, and Statement Number within the corresponding qualifier. Area, Location, and Statement Number are printed in ascending order. Size is printed in descending order.

When the XREFA and DMAP options are combined, all data names in the program are printed for both XREFA and DMAP. If the selected options are XREF and DMAP, only the referenced data names are printed. If only DMAP is selected, only the referenced data names are printed.

Use NOXREF when you do not want to produce any cross-reference.

[Figure 50](#) demonstrates how to specify both the cross-reference (XREF) for referenced data names only and the cross-referencing by area, location, size, and statement number (DMAP).

```
OPTION XREF DMAP  
FILE ARFILE FB 352 5280  
  ACCOUNT 2 182      BILLNGDATE 6 44 NU D  
    BALANCE 5 170 PD 2 E  
LIST ACCOUNT BILLNGDATE BALANCE
```

Figure 50 Example: Specify Cross-References

ABEND Analysis

`OPTION $$DUMP`

This `$$DUMP` keyword is used for debugging. This option disables error handling if the program abends allowing S0C7 (data checks) and S0CB (zero divide exceptions) to be captured by the SPIE/STXIT code. For the indicated program exceptions, the normal error analysis report is generated. For all other program exceptions the system abends immediately at the problem point.

Expand Error Message Information

`OPTION MSGCSECT`

The `MSGCSECT` keyword generates additional error message information, which is printed along with other DYL error messages in your listing.

Premature Sort Termination Without Abend

`OPTION NOSORTAB`

If the `NOSORTAB` keyword is specified, the system generates a return code of 8 when a premature termination of SORT occurs. A premature termination of SORT can occur if your program logic issues a STOP or QUIT before SORT has processed all records. In some installations, a SORT return code of 16 causes an abend.

This keyword overrides the DYLINSTL macro parameter setting.

Generated Object Code Listing

`OPTION $$PCB`

The `$$PCB` keyword lists the generated object code in Assembler language format.

Print Entry Point Address

`OPTION [PRINTEP | NOPRINTEP]`

Use the `PRINTEP` keyword to list the names and corresponding entry point addresses of all subroutines invoked by the VISION:Results program. This option is useful for debugging VISION:Results programs, especially in a VSE environment.

Extended Error Analysis

```
OPTION [PRINTEROR | PRINTERR]  
OPTION EXPRTERR  
OPTION NOEXPRTERR
```

If an interrupt occurs during execution, the PRINTEROR option prints an extended error analysis report displaying work areas, buffers, indexes, and so on, used by the VISION:Results program to assist in the determination of the cause of the interrupt.

The EXPRTERR keyword prints the address of the first byte of each line of a FILE or REFORMAT area in hexadecimal in the left margin of the error analysis output. In addition, the offset from the ending byte for each line is printed in decimal and hexadecimal in the right margin.

To prevent the addresses and offsets from printing, regardless of the DYINSTL setting, use option NOEXPRTERR.

Error analysis prints only if an interrupt occurs during the execution of the VISION:Results program. In [Figure 51](#), PRINTERR is specified during the restoration of the module WKMSTR.

```
OPTION RESTORE WKMSTR DATA '01/15/01' PRINTERR
```

Figure 51 Example: Extended Error Analysis

Verify Syntax

```
OPTION VERIFY
```

To scan VISION:Results statements to verify their syntax without executing them, use the VERIFY keyword. When used in programs that generate multiple reports, OPTION VERIFY applies to all requests.

Options Useful for Technical Support

If you have problems executing your VISION:Results program and need to send the listing to CA Technical Support, execute the program with the following OPTION statements to help resolve the problem quickly:

```
OPTION PRINTEROR XREFA DMAP $$PCB MSGCSECT PRINTEP  
OPTION TEST LIST26 LISTAUD LISTINS
```

Figure 52 Options for Technical Support

Chapter 5: Data Name Qualification

The qualified data name feature (OPTION QLF) allows you to use non-unique data names in your program, as long as they are assigned to different qualifiers. A qualifier is any file name, table name, array name, or named WORKAREA. This option also allows the definition of non-unique, self-defining data names on separate file statements. If you do not use this option, all data names must be unique.

The DYLINSTL parameter QLF=Y can be specified to default to Qualifier mode. For details, see the *Advantage VISION:Results for z/OS Installation Guide*. Any setting in the DYLINSTL macro is overridden by OPTION QLF | NOQLF.

The complete format of the command is:

```
OPTION [QLF|QUALIFIERS|NOQLF|NOQUALIFIERS]
```

Figure 53 OPTION QLF Command Format

The default keyword is NOQLF or NOQUALIFIERS. To use the OPTION, code:

```
OPTION QLF
```

or

```
OPTION QUALIFIERS
```

In addition to making your program easier to read and understand, OPTION QLF is useful if you have a large number of data names in a program and the files, tables, and arrays you are using have similar fields. For example, if you are defining two files that each have an account field, you can code:

```
OPTION QLF

FILE AFIL FB 80 1600
  ACCOUNT 7

FILE BFIL FB 80 1600
  ACCOUNT 7
```

In this case, AFIL and BFIL are considered qualifiers. When the two ACCOUNT fields are referenced in the program, they must be written as AFIL.ACCOUNT and BFIL.ACCOUNT.

Qualification Format

```
qualifier . dataname  
  
      ?  
1-8   2-50  
characters characters
```

Figure 54 Qualification Format

Qualified data names must always be referenced in your program logic in the following manner:

```
OPTION QLF  
  
FILE ARFILE FB 352 5280  
  AMOUNT 5 170 PD 2 A  
  
TABLE ARTBL F 6 ENTRY TBLENT STATUS TBLSTAT  
  AMOUNT 5 PD 2  
.  
MOVE ARFILE.AMOUNT TO ARTBL.AMOUNT
```

For example:

```
BALTOT = ARFILE.AMOUNT + BALTOT  
  
AMTDUE = ARFILE.AMOUNT * BFILE.RATE  
  
SORT ARFILE USING ARFILE.ACCOUNT  
  
LIST ARFILE.ACCOUNT ARFILE.NAME ARFILE.BALANCE  
  
MOVE ARFILE.NAME TO CUSTFILE.NAME
```

Qualifying of Keywords and Self-Defining Data Names

When `OPTION QLF` is specified, you automatically qualify the self-defining data names for the keywords `COUNT`, `LENGTH`, and `STATUS`, whether or not they are specified in the `FILE` statement definition. For example:

```
OPTION QLF  
  
FILE ARFILE FB 352 5280
```

The above `FILE` statement does not specify the keywords `COUNT`, `STATUS`, or `LENGTH`. When you specify `OPTION QLF` or `OPTION QUALIFIERS`, you can qualify these areas as:

```
ARFILE.COUNT  
ARFILE.STATUS  
ARFILE.LENGTH
```

If you specify the `FILE` statement with `COUNT`, `STATUS`, or `LENGTH`, you can use either form of qualification. For example:

```
FILE ARFILE FB 352 5280 COUNT ARCNT  
  
IF ARFILE.ARCNT . . .  
  
MOVE ARFILE.COUNT TO . . .
```

When you define a table in a program, the self-defining data names for the keywords STATUS, ENTRY, KEYLOC, KEYLEN, LENGTH, and OFFSET are also automatically qualified, and, if specified in the TABLE statement, can be qualified using either form. For example:

```
TABLE XTBL F 40 ENTRY TBENT KEYLOC KLOC KEYLEN KLEN
  STATUS TSTAT

MOVE 10 TO XTBL.KEYLOC...

MOVE 1 TO XTBL.TBENT...
```

For arrays, the self-defining data name for the STATUS keyword is automatically qualified.

Qualifying Work Areas

You can also use a named WORKAREA as a qualifier. In this case, the qualifier size must be from two to eight characters long.

In the following example, the WORKAREA named HOLD (from two to eight characters long) is used as a data name qualifier in the program logic:

```
OPTION QUALIFIERS

FILE ARFILE FB 352 5280 STATUS EOFAR
  ACCOUNT 7 4
  LASTNAME 25 85
  BALANCE 5 170 PD 2 A

WORKAREA
  BALTOT 7 PD 2

WORKAREA NAMED HOLD
  LASTNAME 25

IF ARFILE.LASTNAME EQ HOLD.LASTNAME
  BALTOT = BALTOT + BALANCE ENDIF
```

Qualifying with Multiple Reports

If you are using a retained file, table, or array in a multiple-report program, the ID name becomes the qualifier. For more information, see [Chapter 18: Multiple Reports and Multiple Requests](#). In the following example, an array called NUMLIST is retained and uses the array data names in a second report request.

```
FILE NUMFILE FB 80 1600 STATUS NUMSTAT
  NAME 25
  NUM 7 PD

ARRAY NUMLIST F 32 DIM(6 2) STATUS ARSTAT RETAIN
  NAME 25
  NUM 7 PD

MOVE NUMFILE.NAME TO NUMLIST.NAME
MOVE NUMFILE.NUM TO NUMLIST.NUM

REPORT2
```

```
USE NUMLIST  
  
IF NUMLIST.NUM GT 1000  
  MOVE 0 TO NUMLIST.NUM ENDIF
```

MOVE CORRESPONDING and OPTION QLF

You can move several data name areas at the same time by using the MOVE CORRESPONDING command. For more information, see [Chapter 10: Using the MOVE Command](#). The format is:

```
MOVE {CORRESPONDING | CORR} qualifier TO qualifier [SUPPRESS]
```

Figure 55 MOVE CORRESPONDING Command

The fields to be moved must have a different qualifier than the receiving fields, but the same data names. The sending and receiving fields do not have to be of the same data type, but they must conform to the standard rules for moves. For more information, see [Chapter 3: Syntax Rules](#). The following example uses MOVE CORRESPONDING:

```
FILE ARFILE FB 80 STATUS ARSTAT  
  ACCT 2  
  NAME 25  
  BALANCE 7 PD 2 A  
  
FILE CUSTFILE FB 80 STATUS CUSTSTAT  
  NAME 25  
  ADDRESS 15  
  PHONE 10  
  ACCT 2  
  BALANCE 7 PD 2 E  
  
MOVE CORRESPONDING ARFILE TO CUSTFILE
```

In this example, all data name areas from ARFILE having a CORRESPONDING data name area in CUSTFILE are moved.

```
MOVE CORRESPONDING ARFILE TO CUSTFILE
```

This statement does three moves simultaneously:

```
MOVE ARFILE.ACCT TO CUSTFILE.ACCT  
MOVE ARFILE.NAME TO CUSTFILE.NAME  
MOVE ARFILE.BALANCE TO CUSTFILE.BALANCE
```

When you use MOVE CORRESPONDING in your program, the generated program listing shows the individual MOVE statements with the word INSERT printed to the left of each MOVE statement. If you do not want this expansion, code the word SUPPRESS at the end of your MOVE statement. For example:

```
MOVE CORR ARFILE TO CUSTFILE SUPPRESS
```

For more information, see [Chapter 10: Using the MOVE Command](#).

Options XREF/XREFA and DMAP with QLF

When you request the Data name Map (DMAP) and the cross-reference (XREFA or XREF) options in a program, the cross-reference listing is printed with the data names in alphabetical order within the corresponding qualifiers and the data name map is printed in area, location, and size (descending sequence) with the corresponding qualifiers. For more information, see [XREFA Option on page 97](#) and [DMAP Option on page 98](#).

The following examples show the files and work areas of a program and the cross-reference and data name map listings that are generated from them.

```
OPTION DMAP XREFA QLF STRUCTURED ;VERIFY

FILE ARFILE FB 352 5280 STATUS EOFAR
ACCOUNT      7      4
LASTNAME     15     11
FIRSTNAME    10     26
BILLDATE     6      44
NAME         25     85
ADD1         25     110
ADD2         25     135
BALANCE      5     170 PD 2 A
ACCTCODE     2     182
INSTLBAL     6     191 PD 2 E
INSTPAY      3     197 PD 2 A
INTPART      3     206 PD 2 A
NUMPY        2     209 PD  A

FILE ARSUB FB 39 OUTPUT FROM ARSUB STATUS EOFARS
ACCOUNT      7
LASTNAME     15
FIRSTNAME    10
BALANCE      5 PD 2 A
ACCTCODE     2

WORKAREA NAMED WORK1
LASTNAME     15

WORKAREA
BALTOT       7 PD 2 A
BALTOT1      7 PD 2 A
BALTOT2      7 PD 2 A
.
.
(Program logic)
.
.
```

Figure 56 Example Files and Work Areas of a Program

XREFA Option

```
COMPUTER ASSOCIATES VISION:RESULTS DATE 01/15/99
CROSSREFERENCE

A T D
R Y E
E P C
QUALIFIER DATENAME OR TAG LOCN A SIZE E . DEFN REFERENCES
ARFILE ACCOUNT 4 A 7 CH 3
ARSUB ACCOUNT 1 J 7 CH 19
ARFILE ACCTCODE 182 A 2 CH 11
ARSUB ACCTCODE 38 J 2 CH 23- 48
```

Figure 57 Example XREFA Option (Page 1 of 2)

ARFILE	ADD1	110	A	25	CH	8- 36 37
ARFILE	ADD2	135	A	25	CH	9- 37
ARFILE	ARFILE	1	A	1	CH	2
ARSUB	ARSUB	1	J	1	CH	18
ARFILE	BALANCE	170	A	5	PD 2	10- 44 50
ARSUB	BALANCE	33	J	5	PD 2	22- 42 52
	BALTOT	409	R	7	PD 2	31- 44 44 47
	BALTOT1	416	R	7	PD 2	32- 42 42 47 52
	BALTOT2	423	R	7	PD 2	33- 47 50
ARFILE	BILLDATE	44	A	6	CH	6- 35
	C%ARFILE	ARC	U	5	PD 2	
	C%ARSUB	JRC	U	5	PD	18
ARFILE	COUNT	ARC	U	5	PD	2
ARSUB	COUNT	JRC	U	5	PD	18
ARFILE	EOFAR	ARF	U	1	CH	2
ARSUB	EOFARS	JRF	U	1	CH	18
ARFILE	FIRSTNAME	26	A	10	CH	5- 40
ARSUB	FIRSTNAME	23	J	10	CH	21- 40
ARFILE	INSTLBAL	191	A	6	PD 2	12- 49
ARFILE	INSTPAY	197	A	3	PD 2	13- 49
ARFILE	INTPART	206	A	3	PD 2	14
	L%ARFILE	ARS	U	2	BI	2
	L%ARSUB	JRS	U	2	BI	18
ARFILE	LASTNAME	11	A	15	CH	4- 39
ARSUB	LASTNAME	8	J	15	CH	20- 39
WORK1	LASTNAME	393	R	15	CH	27- 48
ARFILE	LENGTH	ARS	U	2	BI	2
ARSUB	LENGTH	JRS	U	2	BI	18
ARFILE	NAME	85	A	25	CH	7- 36
ARFILE	NUMPY	209	A	2	PD 0	15- 49
	S%ARFILE	ARF	U	1	CH	2
	S%ARSUB	JRF	U	1	BI	18
ARFILE	STATUS	ARF	U	1	CH	2
ARSUB	STATUS	JRF	U	1	CH	18
TAG	SUBRT00:					46- 41
TAG	SUBRT99:					51- 41
WORK1	WORK1	393	R	1	CH	26

Figure 57 Example XREFA Option (Page 2 of 2)

If data names containing more than 42 characters are included in your program, the Area and Location sections do not print on the cross-reference listing when XREFA or XREF is requested.

DMAP Option

STERLING SOFTWARE VISION:RESULTS DATE 01/15/99						
DATA NAME MAP						
QUALIFIER	DATENAME OR TAG	LOCN	A	SIZE	T D R Y E E P C E .	DEFN REFERENCES
TAG	SUBRT00:					46- 41
TAG	SUBRT99:					51- 41
ARFILE	ARFILE	1	A	1	CH	2
ARFILE	ACCOUNT	4	A	7	CH	3
ARFILE	LASTNAME	11	A	15	CH	4- 39
ARFILE	FIRSTNAME	26	A	10	CH	5- 40
ARFILE	BILLDATE	44	A	6	CH	6- 35
ARFILE	NAME	85	A	25	CH	7- 36
ARFILE	ADD1	110	A	25	CH	8- 36 37
ARFILE	ADD2	135	A	25	CH	9- 37
ARFILE	BALANCE	170	A	5	PD 2	10- 44 50
ARFILE	ACCTCODE	182	A	2	CH	11
ARFILE	INSTLBAL	191	A	6	PD 2	12- 49
ARFILE	INSTPAY	197	A	3	PD 2	13- 49
ARFILE	INTPART	206	A	3	PD 2	14
ARFILE	NUMPY	209	A	2	PD 0	15- 49

Figure 58 Example DMAP Option (Page 1 of 2)

ARSUB	ACCOUNT	1	J	7	CH	19		
ARSUB	ARSUB	1	J	1	CH	18		
ARSUB	LASTNAME	8	J	15	CH	20-	39	
ARSUB	FIRSTNAME	23	J	10	CH	21-	40	
ARSUB	BALANCE	33	J	5	PD	22-	42	52
ARSUB	ACCTCODE	38	J	2	CH	23-	48	
WORK1	LASTNAME	393	R	15	CH	27-	48	
WORK1	WORK1	393	R	1	CH	26		
	BALTOT	409	R	7	PD	31-	44	44 47
	BALTOT1	416	R	7	PD	32-	42	42 47 52
	BALTOT2	423	R	7	PD	33-	47	50
ARFILE	COUNT	ARC	U	5	PD	2		
	C%ARFILE	ARC	U	5	PD	2		
ARFILE	EOFAR	ARF	U	1	CH	2		
	S%ARFILE	ARF	U	1	CH	2		
ARFILE	STATUS	ARF	U	1	CH	2		
ARFILE	LENGTH	ARS	U	2	BI	2		
	L%ARFILE	ARS	U	2	BI	2		
	C%ARSUB	JRC	U	5	PD	18		
ARSUB	COUNT	JRC	U	5	PD	18		
	S%ARSUB	JRF	U	1	BI	18		
ARSUB	EOFARS	JRF	U	1	CH	18		
ARSUB	STATUS	JRF	U	1	CH	18		
ARSUB	LENGTH	JRS	U	2	BI	18		
	L%ARSUB	JRS	U	2	BI	18		

Figure 58 Example DMAP Option (Page 2 of 2)

No Qualified Data Names Option

When you code `OPTION QLF` or `OPTION QUALIFIERS` at the beginning of your program, it remains in effect for the entire program. In a multiple-report run, this option remains in effect for all requests, unless you code `OPTION NOQLF` or `OPTION NOQUALIFIERS` in a subsequent request. `OPTION NOQLF` turns off the qualified data name option.

You cannot use both `OPTION QLF` and `OPTION NOQLF` in a single report request, but you can change modes between requests in a multiple-report run.

Chapter 6: Using the FILE Command

A FILE statement must be coded to define every file to be read or written by VISION:Results. The FILE statements typically are the first statements to appear, unless preceded by an OPTION statement.

```
OPTION HEXPRINT FILEIN
FILE FILEIN FB 80 800
FILE FILE01 OUTPUT FROM FILEIN FB 80 4000
FILE FILE02 OUTPUT FROM FILEIN FB 80 4000
```

VISION:Results directly supports up to eight input (seven when SORT is used) and eight output files in a run. The files can be of any supported organization (sequential, BDAM, [ISAM](#), or VSAM) and format (fixed, variable, variable-spanned, or undefined) in any combination.

You can process an unlimited number of sequential input and output files using the EXIT module, DYLIOU. The following file organizations and formats are supported by standard file processing:

Sequential	Fixed, variable, variable-spanned (z/OS only), and undefined formats.
BDAM (or DAM)	Fixed, variable, and undefined formats.
ISAM	Fixed, variable (z/OS only) formats.
VSAM KSDS, ESDS, and RRDS	Fixed, variable format for KSDS and ESDS, and fixed for RRDS.
PDS	Fixed, variable, and undefined formats.
SSL	Fixed format.
VISION:Results EXIT facility	Fixed, variable, variable-spanned (z/OS only), and undefined formats. Same as sequential organization.
Database support	IMS (optional interface) DL/I (optional interface) IDMS/DB (optional interface) DB2 (optional interface) DBC (optional interface) ADABAS, DATACOM/DB, TOTAL (using CALL command)

The following I/O modes are supported:

- Sequential organization, including:
 - Sequential retrieval
 - Sequential write
- BDAM (or DAM) organization, including:
 - Sequential retrieval
 - Sequential load
 - Random retrieval
 - Random retrieval and update
- [ISAM](#) organization, including:
 - Sequential retrieval
 - Sequential load/load extend
 - Random retrieval
 - Random retrieval and update
 - Limited sequential (SETL) retrieval by full or generic key
- VSAM organization, including:
 - Sequential retrieval
 - Sequential load/load extend/insert
 - Random retrieval
 - Random retrieval and update
 - Sequential retrieval and update
 - Limited sequential (skip) retrieval by full or generic key
- PDS organization, including:
 - Sequential member retrieval
 - Selective member retrieval
 - Full directory retrieval
 - Sequential member write
 - Sequential directory write
- SSL organization, including:
 - Sequential retrieval
 - Selective retrieval
- EXIT organization, including:
 - Sequential retrieval
 - Sequential write

FILE statements are not required for the following files:

- VISION:Results source program listing file and file print output, which is SYSLST (VSE) or SYSPRINT (z/OS and VM)
- VISION:Results report file, which is usually SYSLST (VSE) or SYS280R (z/OS and VM)
- SORT files for work, library (z/OS SORTLIB), and print (z/OS SYSOUT)
- VISION:Results object file from freeze run, which is SYSPCH (VSE) or SYS280FZ (z/OS and VM)
- VISION:Results work files, which are SYS004, AUDEPF, AUDWRK, AUDPRINT, and AUDCBF (z/OS and VM) or IJSYS04, SYS008, SYS009, SYS010, and SYS011 (VSE)
- Database or other user files to be read, written, or updated using the CALL facility
- JCL must be included for all files processed, even if a file is not defined in a FILE statement

The rules regarding FILE statements are:

- Input files can be defined anywhere in the program and must be defined before they are referenced.
- Output files that are output from themselves can be defined anywhere in the program and must be defined before they are referenced.
- Output files that are output from another file must be defined before any processing phase keywords are defined.
- Coding self-defining data names that are a subset of the file name is allowed.

Supported Access Methods

The following notation rules describe the syntax of FILE statements:

- A vertical bar | separates mutually exclusive alternatives.
- Underlined text shows default options.
- Braces { } show related alternative items. One item must be selected.
- Brackets [] indicate optional items.
- Uppercase letters, words, and special characters must be specified exactly as shown.
- Lowercase letters, words, and special characters require a specific value to be substituted in their place.
- Parentheses must be typed exactly as shown.
- Keywords and operands between a set of ! and !... indicate that the repetition of what is between the ! and !... is allowed.
- Bold text indicates VSE only.

The required syntax for the FILE statement is shown below:

BDAM	Chapter 6: Using the FILE Command and Chapter 22: BDAM Processing
ISAM	Chapter 6: Using the FILE Command and Chapter 21: ISAM Processing
SEQUENTIAL	Chapter 6: Using the FILE Command
VSAM ESDS	Chapter 6: Using the FILE Command and Chapter 21: ISAM Processing
VSAM KSDS	Chapter 6: Using the FILE Command and Chapter 21: ISAM Processing
VSAM RRDS	Chapter 6: Using the FILE Command and Chapter 21: ISAM Processing
EXIT	Chapter 6: Using the FILE Command
PDS	Chapter 6: Using the FILE Command and Chapter 23: Using PDS and SSL Support
SSL	Chapter 6: Using the FILE Command and Chapter 23: Using PDS and SSL Support

A detailed explanation for each keyword and command can be found at the beginning of the FILE syntax illustrations.

BDAM (or DAM) Files

For more information on BDAM processing, see [Chapter 23: Using PDS and SSL Support](#).

```
FILE filename
BDAM {RELTRK dataname2 | RELBLK dataname2}
STATUS dataname2
{F | V | U} recordsize
{INPUT | OUTPUT FROM filename | IO}
{RANDOM dataname2 {CH | NU | BI | PD} [decimals] |
  SEQUENTIAL [dataname2 {CH | NU | BI | PD} [decimals]]}
[KEYLEN nnn] [COUNT dataname2] [LENGTH dataname2] [DYNAM]
[DISK {2311 | 2314 | 3330 | 3340 | 3350}]
[SYSnnn] [EXTENTS nn]
```

Figure 59 BDAM Processing

```
FILE DAFILE BDAM OUTPUT FROM DAFILE F 35
RANDOM RELBLK BLKNUM STATUS BDAMSTAT
```


EXIT Files

For detailed information on writing an EXIT module, see [Chapter 13: Using the CALL Command](#).

```
FILE filename
EXIT modulename [nK]
[SEQUENTIAL]
[{INPUT | OUTPUT FROM filename | IO}]
[{FB | VB | S | F | V | U | SB} [recordsize [blocksize]]]
[SIZE nnnnn]
[PARM ({WORK | dataname3 ...})]
[STATUS dataname2] [DYNAM]
[TAPE] [SYSnnn] [DISK {2311 | 2314 | 3330 | 3340 | 3350 |
3375 | 3380 | FBA}]
```

Figure 60 EXIT Module

```
FILE TEMPFL VB 75 1508 OUTPUT FROM TEMPFL
EXIT DYLI0U
```

ISAM Files

For more information on [ISAM](#) processing, see [Chapter 21: ISAM Processing](#).

```
FILE filename
ISAM
{INPUT | OUTPUT FROM filename | IO}
[{FB | VB | F | V} [recordsize [blocksize]]]
[RANDOM dataname2 {CH | NU | BI | PD} [decimals] |
SKIP dataname2 {CH | NU | BI | PD} [decimals] | SEQUENTIAL}
[KEYLEN nnn [KEYLOC nnnn]] [POSITION dataname2]
[STATUS dataname2] [COUNT dataname2] [LENGTH dataname2]
[BYPASS nn] [DROPERR] [DUMMY] [NULL]
[RETAIN | NORETAIN]
[{MODIFY modulename [nK]} [SIZE nnnnn]]
[PARM ({IOAREA | WORK | dataname3...})] [DYNAM]
[DISK {2311 | 2314 | 3330 | 3340}]
[EXTENTS nn] [MASTER] [CYLOFL nn]
[SYSnnn] [ONEBUFF]
```

Figure 61 ISAM Processing

```
FILE OUTFILE ISAM OUTPUT FROM INFILE FB 120 1200
  KEYLEN 3 KEYLOC 1
FILE ISAMFL ISAM FB 70 700 RANDOM SKEY
  KEYLEN 3 KEYLOC 1 STATUS ISTAT
FILE ISAMFIL ISAM FB 70 700 DISK 3340 RANDOM SKEY
  KEYLEN 3 KEYLOC 1
```

PDS Files

For more information on PDS syntax, see [Chapter 23: Using PDS and SSL Support](#).

```
FILE filename
PDS dataname MEMBER dataname [NEWNAME dataname] STATUS dataname
[INPUT | OUTPUT FROM filename]
[{FB | VB | F | V | U} [recordsize [blocksize]]] [DYNAM]
[LENGTH dataname]
[COUNT dataname]
```

Figure 62 PDS Syntax

```
FILE PDSOURCE PDS DSNAREA MEMBER MEMNAME COUNT MEMCOUNT
  STATUS PDSTATUS
FILE SYSOURCE PDS DSNAREA U MEMBER SYSNAME
  LENGTH SYSNAME STATUS SYSTATUS
```

SEQUENTIAL Files

```

FILE filename
[SEQUENTIAL]
{INPUT | OUTPUT FROM filename}
[{FB | VB | S | F | V | U | SB} [recordsize [blocksize]]]
[{CARD | CARDS} [{INTERPRET | INTSEL | SELECT | JCL}] ]
[STATUS dataname2] [COUNT dataname2] [LENGTH dataname2]
[BYPASS nn] [DROPER] [DUMMY] [NULL]
[RETAIN | NORETAIN]
[{MODIFY modulename [nK] } [SIZE nnnnn]
  [PARM ({IOAREA | WORK | dataname3...})] ] [DYNAM]
[DISK {2311 | 2314 | 3330 | 3340 | 3350 | 3375 | 3380 | FBA}]
[DISKETTE | {DATAVER | NOFEED | PROTECT | SECURE | VERIFY |
  VOLVER}] [NUMFILES dataname2]
[SYSnnn] [ONEBUFF]
[TAPE [{NL | SL | T}] [{REWIND | NORWD | UNLOAD}] [n]
[NUMFILES dataname2]]

```

Figure 63 SEQUENTIAL Files

```

FILE ARFILE
FILE OUTFILE FROM OUTFILE OUTPUT FB 25 2500
FILE ARFILE FB 352 5280 DISK 3330 SYS027

```

SSL Files

For more information on SSL syntax, see [Chapter 23: Using PDS and SSL Support](#).

```

FILE filename
SSL BOOK dataname2 STATUS dataname2
[COUNT dataname2]

```

Figure 64 SSL Syntax

```

FILE ARTEXT BOOK BOOKNAME STATUS TEXTSTAT
FILE ARTEXT BOOK SOURCEPGM STATUS SOURCESTAT
COUNT SOURCECNT

```

VSAM ESDS Files

For more information about the keywords and operands used by VSAM ESDS file, see [Chapter 20: VSAM Processing](#).

Note: In the example below, the **bold** text is optional if you have used the OPTION VSAMCAT statement or the VSAMCAT=Y keyword in the DYLINSTL macro.

When using this feature, you do not need to specify the file type and its characteristics, just type VSAM in the FILE statement. The VSAMCAT option must be in effect to process RLS files correctly.

For examples of these programs, see [Chapter 21: ISAM Processing](#).

```
FILE filename [VSAM]
ESDS [PASSWORD {'password' | dataname2}]
{INPUT | OUTPUT FROM filename | IO}
[F | V | S] [recordsize]
{RANDOM dataname2 | RANDOMX dataname2} |
  {SKIP dataname2 | SKIPX dataname2} | SEQUENTIAL}
[{RELBYTE dataname2 | RELBYTEX dataname2}] [POSITION dataname2]
[STATUS dataname2] [COUNT dataname2] [LENGTH dataname2]
[DROPPERR] [DUMMY] [REUSE | NULL]
[RETAIN | NORETAIN]
[{MODIFY modulename [nK}] [SIZE nnnnn]
  [PARM ({IOAREA | WORK | dataname3...})]] [DYNAM]
[SYSnnn]
```

Figure 65 VSAM ESDS Files

```
FILE TRANS ESDS F 40
FILE OUTFILE ESDS V 210 LENGTH OUTLEN
  OUTPUT FROM OUTFILE
```

VSAM KSDS Files

For more information about the keywords and operands used by VSAM KSDS files, see [Chapter 20: VSAM Processing](#).

Note: In the example below, the **bold** text is optional if you used the OPTION VSAMCAT statement or the VSAMCAT=Y keyword in the DYLINSTL macro.

When using this feature, you do not need to specify the file type and its characteristics. Type VSAM in the FILE statement. The VSAMCAT option must be in effect to process RLS files correctly.

```
FILE filename [VSAM]
KSDS [PASSWORD {'password' | dataname2}]
{INPUT | OUTPUT FROM filename | IO}
[{F | V | S} recordsize]
{RANDOM dataname2 {CH | NU | BI | PD} [decimals] |
  SKIP dataname2 {CH | NU | BI | PD} [decimals] | SEQUENTIAL
[KEYLEN nnn [KEYLOC nnnn]] [POSITION dataname2]
[REUSE | NULL] [ERASE dataname2] [PARTKEY nnn]
[STATUS dataname2] [COUNT dataname2] [LENGTH dataname2]
[DROPERR] [DUMMY]
[RETAIN | NORETAIN]
[{MODIFY modulename [nK] } [SIZE nnnnn]
  [PARM ({IOAREA | WORK | dataname3...})]] [DYNAM]
[SYSnnn]
```

Figure 66 VSAM KSDS Files

```
FILE FKSDSI KSDS F
FILE FKSDS KSDS OUTPUT FROM SEQIN F
FILE VSAMIN KSDS V COUNT KCOUNT LENGTH KLENGTH
```

VSAM RRDS Files

For more information about the keywords and operands used by VSAM RRDS files, see [Chapter 20: VSAM Processing](#).

Note: In the example below, the **bold** text is optional if you have used the OPTION VSAMCAT statement or the VSAMCAT=Y keyword in the DYLINSTL macro.

When using this feature, you do not need to specify the file type and its characteristics, just type VSAM in the FILE statement. The VSAMCAT option must be in effect to process RLS files correctly.

For examples of these programs, see [Chapter 21: ISAM Processing](#).

```
FILE filename [VSAM]
RRDS [PASSWORD {'password' | dataname2}]
{INPUT | OUTPUT FROM filename | IO}
[{F} [recordsize]]
{RANDOM dataname2 {CH | NU | BI | PD} [decimals] |
  SKIP dataname2 {CH | NU | BI | PD} [decimals] | SEQUENTIAL}
[RELREC dataname2] [POSITION dataname2] [ERASE dataname2]
[STATUS dataname2] [COUNT dataname2] [LENGTH dataname2]
[DROPERR] [DUMMY] [REUSE | NULL]
[RETAIN | NORETAIN]
[{MODIFY modulename [nK]} [SIZE nnnnn]
  [PARM ({IOAREA | WORK | dataname3...})]] [DYNAM]
[SYSnnn]
```

Figure 67 VSAM RRDS Files

```
FILE VRRDS1 RRDS F 40 OUTPUT FROM FILEIN
RELREC RELRECNUM
FILE EXAMPLE RRDS F 108 IO RANDOM RELRCD
STATUS STATFLG
```

Syntax Definitions (General)

Filename

A file name is required. It must immediately follow the FILE command, and you can specify any valid name. A z/OS or VM file name can be from one to eight characters long and must start with an alphabetic character. This file name also serves as the ddname in the z/OS JCL and FILEDEF in VM.

A VSE file name can be from one to seven characters long and is the file name that appears in the DLBL or TLBL JCL statement for the file.

```
FILE ACCTMST ...
```

The JCL looks like this:

```
z/OS:          //ACCTMST DD DSN=T01.MYDS,DISP=SHR
VSE:           // ASSGN SYSnnn,X'131'
               // DLBL ACCTMST,'MASTERFILE' ...
               // EXTENT SYSnnn ...
```

File Organization

File organization is required if it is not physical sequential. The supported values are as follows:

SEQUENTIAL	Default
ISAM	Indexed sequential
ESDS	VSAM entry sequence
KSDS	VSAM keyed sequence
RRDS	VSAM relative record sequence
BDAM	Direct access
PDS	Partitioned data set

```
FILE ISFILE ISAM ...
```

Input/Output Type

An entry is required if not INPUT. This specifies whether the file is to be opened as input, output, or both. The valid values are:

INPUT	Default. The file is opened as an input file. This is the case if you are processing the file in sequential, random retrieve, or skip sequential mode.
OUTPUT	The file is opened as an output file. This is coded when you are creating a sequential, ISAM , or VSAM file. You can also code this if you are load-extending an ISAM or VSAM file, or sequentially inserting a new record in a VSAM KSDS file.
I/O	The file is both read from and written to. This is coded if you are randomly reading and writing an ISAM or VSAM file or if you are sequentially reading and updating a VSAM file.

Processing Mode

The processing mode is not required if the sequential, [ISAM](#), or VSAM file is being read sequentially from the beginning. You must specify the processing mode if it is to be random, skip sequential [ISAM](#), or VSAM. The supported values are:

- [SEQUENTIAL on page 111](#)
- [RANDOM dataname \(data type\) and RANDOMX dataname on page 127](#)
- [SKIP dataname \(data type\) and SKIPX dataname on page 128](#)

SEQUENTIAL

Default. The file is read from the beginning or is created. You also code this if you are load-extending an [ISAM](#) or VSAM file, or inserting new records into a VSAM KSDS file.

```
FILE FILEIN FB 80 800
```

The default values for file organization (SEQUENTIAL), I/O type (INPUT), and processing mode (SEQUENTIAL) are assumed.

```
FILE FILEIN SEQUENTIAL INPUT SEQUENTIAL FB 80 800
```

This defines the same file that is shown in the previous example, except the default values have been coded.

```
FILE FILEIN KSDS IO SEQUENTIAL F 124 KEYLEN 8
```

A KSDS VSAM file is read sequentially from the beginning; its records are updated or deleted, or new records are inserted.

FROM filename

This is required if a sequential, [ISAM](#), or VSAM output file is being created/loaded. This is optional if a random [ISAM](#) or VSAM I/O file is being defined.

The file area from which the record is written is identified by the FROM keyword. Following FROM, code the name of the file in which the area holds the record to be written. This can be an input file record, or the record can be written from its own output hold area, provided you have moved the record to this area yourself during the program logic.

If you do not code this phrase while defining a random I/O file, VISION:Results assumes that you are writing from its own input area.

```
FILE FILEIN INPUT FB 80 1600
FILE FILEOUT OUTPUT FROM FILEIN FB 80 4000
```

The output file FILEOUT is being copied from the input file FILEIN.

```
FILE ACCTIN INPUT FB 80 1600
  NAMEI 30 ACCOUNTI 5 50
FILE ACCTNEW OUTPUT FROM ACCTNEW FB 120 2400
  ACCOUNTO 5 NAMEO 30 ...
  .
  .
  .
MOVE NAMEI TO NAMEO MOVE ACCOUNTI TO ACCOUNTO
```

An entirely new record is formatted using some of the fields from the input file. The record is built in the output record's work area, so VISION:Results is told to write from this area.

Record Format

Record format is required as follows:

- For all VSE files
- For all BDAM files
- For all z/OS VSAM files, unless the VSAMCAT option is in effect and the DD statement for the corresponding VSAM file is included in the JCL

The acceptable values are:

blank	The record format is obtained from the JCL or the data set label (z/OS).
F	The record format is fixed unblocked.
V	The record format is variable unblocked.
S	The record format is variable-spanned unblocked (z/OS only for sequential files; z/OS and VSE for VSAM KSDS and ESDS files).
U	The record format is undefined.
FB	The record format is fixed blocked.
VB	The record format is variable blocked.
SB	The record format is variable-spanned blocked (z/OS only).

Record Size

Code the record size attribute of the file you are defining under the following conditions:

For VSE:	■ Sequential files ■ ISAM files ■ BDAM files ■ VSAM files when the processing mode is I/O ■ Sorted files
For z/OS:	■ Input files that are unlabeled and do not have LRECL coded in the JCL DD statement ■ Output files that do not have LRECL coded in the JCL DD statement ■ ISAM files when the processing mode is I/O ■ BDAM files ■ VSAM files when the processing mode is I/O

The record size value can range from 1 to 32767 (32752 in the case of variable-length files). If defining a fixed-length file, code the fixed-length size. If defining a variable-length file, code only the maximum length of the data; do not include the length of the Record Descriptor Word (RDW) that always precedes the data portion of a variable-length non-VSAM record. A VSAM record does not have an RDW. Exception: If exit DYLI0U is used, then the 4 bytes for the RDW should be included in the maximum record size. If defining an undefined format file, code the length of the longest possible record (block) on the file.

If you are coding the record size, you must immediately precede it with the record format.

```
FILE FIXIN INPUT F 80
```

Here, a fixed unblocked input file is being defined. Each record is 80 bytes long.

```
FILE VAROUT OUTPUT VB 80 844 FROM VARIN ...
```

A variable-length file is created. The data portion of each record has a maximum length of 80 bytes. VISION:Results adds 4 more bytes for the RDW when setting the DCB/DTF attributes. The maximum block size, including the length of the RDW fields and the Block Descriptor Word (BDW) field, is 844.

```
FILE MYVSAM KSDS INPUT F 134
  FIELDX 5 30
  .
  .
  SORT MYVSAM USING FIELDX
```

Above, the record type and length is coded for the VSAM file because it is sorted in the program and SORT needs the length of the record.

Block Size

The block size attribute of the file you are defining must be coded under the following conditions:

- If VSE, and you are defining a fixed or variable blocked sequentially organized file.
- If VSE, and you are defining a fixed blocked [ISAM](#) file.
- If z/OS input, and the file is unlabeled and you have not coded the size in the JCL DD statement of the file.
- If z/OS output, and you have not coded the size in the JCL. If the file is a non-VSAM output file, SYSBLOK was selected in the DYLINSTL macro, and no JCL block size was specified, then the operating system calculates the block size and the block size is not required on the FILE statement.

The block size value can range from 1 to 32767. When specifying the block size of a variable-length file, include the maximum length for the data portion of all records, the RDW fields that precede each record, plus the 4-byte Block Descriptor Word. The maximum block size for variable blocked (VB) is 32760.

When coding the block size in a FILE statement, it must be preceded by the record format and record size.

```
FILE FILEIN INPUT FB 80 1600
```

In this example, the input file is fixed blocked format with a record length of 80 and block size of 1600.

STATUS dataname

This is required for BDAM. One use of this phrase is to allow you to test whether a sequentially read input file has reached end of file. After the keyword STATUS, code a valid data name to identify the STATUS field. You do not have to define the field anywhere because VISION:Results defines the field automatically.

During program execution, when VISION:Results detects the end of file on an input file it is reading, it moves an E to the STATUS field. Additionally, it moves high values (all binary ones) to the file's record area. If all of the sequentially read input files in the program are at end of file, a termination procedure is initiated.

This means that termination procedures are initiated for programs with only one input file, which immediately goes to end of file (a null or empty file). This bypasses any user logic coded to test the status flag or an input field for high values. In this case, the ONE END OF INPUT section must be used to perform any user logic on end of file.

If you are processing multiple input files, check when any one of the files is completed. You can do this by testing the STATUS field for E.

```
FILE SEQIN INPUT FB 140 1400 SYS011
  STATUS STATSEQ
FILE VSAMIN KSDS INPUT RANDOM FINDKEY F 40
  KEYLEN 5 STATUS STATUS
IF STATSEQ EQ 'E' STOP ENDIF
.
.
.
```

Checking the status on the sequential input file is important because the random VSAM file will never reach the end of file and VISION:Results will not terminate the run for you.

For more information about [ISAM](#) and VSAM processing, see [STATUS dataname on page 125](#).

LENGTH dataname

This phrase gives you access to the length of the current record being processed. Follow the LENGTH keyword with a valid data name. You do not have to define this field anywhere because VISION:Results does it automatically.

This is usually an optional phrase, but it is required under the following conditions:

- You have a variable-length file, OUTPUT from itself.
- You have an undefined file, OUTPUT from itself.

Rules for the FILE LENGTH keyword:

- It is not allowed when OUTPUT is specified on the FILE statement, unless it is FROM itself.
- LENGTH must be used on an OUTPUT file output from itself, if the file is variable length. Otherwise, VISION:Results writes out only a 4-byte record. No syntax check is made if LENGTH is not used.

When a record is read from a file, the length of the record just read is placed in the field identified by the data name. This is particularly useful if your file is undefined or variable in format and you need to know the current length. The value in the length field for a variable-length record is the data length only. It does not include the 4 bytes for the RDW unless exit DYLIUO is used.

If you do not tell VISION:Results, using the format coded in the FILE statement, that the file is variable or variable-spanned, the value of the LENGTH field includes the RDW length. If the file is VSAM variable format, the records have no RDW.

```
FILE ACCTREC INPUT VB 300 3044 LENGTH RECLEN
  INACCOUNT 5 7 INNAME 30 45 INBAL 8 98 NU 2
  IF RECLEN EQ 100 GOTO REC100 ENDIF
  IF RECLEN EQ 250 GOTO REC250 ENDIF
  .
  .
  .
```

To process the record, the length of the data portion of the variable-length record must be known.

```
FILE ACCTREC INPUT FB 300 3000
  INACCOUNT 5 1 INNAME 20 15 INBAL 8 35 NU 2
FILE ACCTVAR OUTPUT FROM ACCTVAR VB 300 3044 LENGTH VARLEN
  ACCOUNT 5 1 NAME 30 6 BALANCE 5 36 PD 2
  .
  .
  MOVE INACCOUNT TO ACCOUNT
  MOVE INNAME TO NAME
  MOVE INBAL TO BALANCE
  MOVE 300 TO VARLEN
  .
  .
```

A variable-length file is created from a fixed-length file. You must have a means to establish the length of the record to write. This is done by specifying the LENGTH phrase in the output FILE statement, and then moving the record length value you want to the field identified by the data name. Use the same approach if creating an undefined format file from fixed input.

COUNT dataname

This phrase gives you access to the current record count of the file being read or written. If you code the keyword COUNT followed by a data name of your selection, you can interrogate the record count field in your program logic. This is particularly useful when you want to do either of the following:

- Limit the reading or writing of records.
- Select certain records based on their position in a file.

You do not have to define the record count field, just give it a name. VISION:Results allocates the field automatically. When a record is read or written either as part of the automatic cycle or when you issue a READ or WRITE command, VISION:Results adds 1 to the appropriate record count field.

```
FILE ACCTFLE INPUT FB 100 1000 COUNT INCOUNT
PRINT ACCTFLE
IF INCOUNT EQ 100 STOP ENDIF
```

Records from ACCTFLE are read and printed until the record count reaches 100.

Then, the run is terminated.

```
FILE INFILE
FILE OUTFILE OUTPUT FROM INFILE FB 200 2000 COUNT OUTCOUNT
  IF OUTCOUNT EQ 500 STOP ENDIF
```

An output file is being created directly from input records. In this example, once 500 records have been written, the program is terminated.

Do not alter the contents of the record count field.

CARD or CARDS

The keyword CARD or CARDS in a FILE statement defining an input file specifies that the input is coming from data records following the FIN statement in the program stream. FIN cannot be used within code that is input to VISION:Results using the COPY or COPYE command.

In z/OS, the file name in the FILE statement must be SYSIN; if it is not, VISION:Results changes the file name to SYSIN. In VSE, the file name must be SYSIPT. The JCL ddname of SYSIN accepts input for the non-SYSIN file name after the FIN statement. This file name must not have a z/OS DD JCL statement.

VISION:Results logically reads all data from SYSIN, then separates it into two files. If a READ statement is issued, use the coded file name.

z/OS:

```
FILE SYSIN CARDS
PRINT SYSIN
FIN
DATA RECORD 1
DATA RECORD 2
DATA RECORD 3
```

VSE:

```
FILE SYSIPT CARDS
PRINT SYSIPT
FIN
DATA RECORD 1
DATA RECORD 2
DATA RECORD 3
```

In both the z/OS and VSE examples, instream data to print is performed.

In VSE, you code CARD or CARDS for output directed to SYSPCH.

```
FILE SYSIPT CARD
FILE SYSPCH CARD OUTPUT FROM SYSIPT
FIN
DATA RECORD 1
DATA RECORD 2
```

A VSE VISION:Results program reads data input from the program stream and directs it to SYSPCH.

JCL

VSE only. This keyword can be used along with CARD or CARDS to indicate that instream data containing JCL is being read. This allows //, /*, and /& statements to be read as data and not treated as JCL.

If the JCL keyword is being used, the file name must not be SYSIPT and a SYS number for a programmer logical unit (for example, SYS011) must be specified.

Additionally, you must provide your own delimiter to know when you have read all of the data because /* and /& no longer serve that purpose.

```
// ASSGN SYS011,X'05C'  
// EXEC DYL280,SIZE=250K  
FILE MYDATA CARD JCL SYS011  
  FIELD1 10  
  IF FIELD1 EQ '**END DATA' STOP ENDIF  
  PRINT MYDATA  
  FIN  
  // JOB JOBPROD  
  // EXEC PRODPGM  
  /*  
  /&  
  **END DATA  
  /&
```

This VSE run reads instream data containing JCL and prints each record. When the statement containing the value **END DATA is read, the run is terminated.

INTERPRET

VSE only. If CARD OUTPUT is specified and the punch is a 3525 with the interpret feature, you can code the keyword INTERPRET in your FILE statement to perform the standard 2-line interpret on records being punched.

```
FILE TRANSIN INPUT FB 80 800 TAPE SYS011  
FILE SYSPCH OUTPUT CARD INTERPRET FROM TRANSIN
```

A tape file is read and its records are being punched/interpreted.

SELECT dataname

VSE only. If CARD OUTPUT has been specified and you want to direct punched cards to either stacker 1 or stacker 2, code the keyword SELECT followed by a valid data name to identify the field that contains the stacker select code. You do not have to define this field anywhere. It is done automatically.

Valid stacker select codes are:

- V = Stacker 1
- W = Stacker 2

Before you punch a card, move one of the codes to the data name you selected. The card is then routed to the proper stacker.

```
FILE FILEIN FB 80 1600 DISK 3330 SYS014
  IN1 1 34
FILE SYSPCH OUTPUT CARD SELECT STACKNO FROM FILEIN
  IF IN1 EQ '1' NEXT ELSE GOTO PROC2 ENDIF
  MOVE 'V' TO STACKNO
  WRITE SYSPCH ACCEPT
PROC2:
.
```

Records are read from a tape file. If the IN1 field in the input record is equal to 1, the record is punched and routed to stacker 1. If the IN1 field is not equal to 1, another procedure is performed.

INTSEL dataname

VSE only. If CARD OUTPUT is specified and the punch is a 3525 with interpretation, you can code the keyword INTSEL followed by a data name you specify to allow for the standard 2-line interpretation to occur, and also to direct the punched card to stacker 1 or stacker 2. Either V or W must be moved to the data name prior to punching the card. See [SELECT dataname on page 118](#).

BYPASS nn

z/OS and VSE. Applies to processing. Coding the keyword BYPASS followed by 1 to 99 tells VISION:Results to bypass 1 to 99 blocks of data at the start of the file before processing the first record.

This feature is useful for handling unconventional files, such as 1401 standard labeled tapes, whose data blocks are preceded by other than data records.

```
FILE TAPEIN INPUT FB 100 1000 TAPE NL BYPASS 1 SYS011
PRINT TAPEIN
```

A 1401 labeled tape is read. Because there is no tape mark between the label and the data, the BYPASS option is coded to skip over the 80-byte label.

DUMMY

The keyword DUMMY coded in your FILE statement is useful if you have to do some processing where you do not want VISION:Results to read a file, but need to satisfy the VISION:Results requirement that at least one file is defined. This can be the case if you are either reading a file using an external routine invoked by the CALL statement, or using VISION:Results to test and verify one of your own routines.

Doing a READ to a file having a DUMMY option generates an error of FILENAME NOT DEFINED.

You must terminate the run yourself using the STOP command when you are finished processing. VISION:Results terminates automatically only when sequentially processed files (no random files) have been defined and when all of these files have reached end of file. A DUMMY status creates a condition where the file is on hold and never finishes.

```
FILE SYSIN CARD DUMMY
WORKAREA
  JULDATE 3 PD VALUE 96015
  GREGDATE 8
CALL DATECONV 2K USING JULDATE GREGDATE
PRINT JULDATE LENGTH 11
STOP
FIN
```

VISION:Results is testing the proper execution of a date conversion routine. The DUMMY keyword prevents reading the SYSIN file because no input records are required. The external routine is invoked once using CALL to convert a 3-byte packed Julian date to an 8-byte edited Gregorian date. The date hold areas are printed after the conversion. The run is then terminated. If the STOP command is not included, a validation error is generated.

NULL

z/OS and VSE. This applies to sequential and [ISAM](#) output files only. If there is a possibility that the output file you are defining is not written to, and you want a null file created (a file with labels but no data), code NULL in your FILE statement.

```
FILE SEQOUT OUTPUT FROM XYZ FB 100 1000 SYS021 NULL
```

Above, a sequential output file is defined. To ensure that the file is at least opened and closed to create labels, the NULL option is coded.

The compiler can be customized to provide this facility as a default. For information about the DYLINSTL macro parameter OUTFILE, see the *Advantage VISION:Results for z/OS Installation Guide*.

DROPERR

z/OS and VSE. Input only. If you have an input file and you are abending because of I/O errors on one or more of the blocks on the file, you can often process the file by specifying DROPERR in the FILE statement. When the operating system detects an input error, it drops the block rather than abending. If you use this option, you are losing one or more records depending on how many there are per block, but this is often preferable to not being able to process at all.

VISION:Results lets you know how many blocks it had to drop by printing a dropped block count in the I/O control totals at the end of the run.

```
FILE SEQIN INPUT FB 400 4000 SYS011 TAPE DROPERR
```

A tape file that has bad blocks is read and any blocks with an error are dropped.

SYSnnn

VSE only. This entry is required if the file is on tape. You must specify the SYS number of a Programmer Logical Unit by which the file is identified.

```
FILE SEQIN FB 400 4000 SYS012 TAPE
```

A tape input file is defined and its SYS number is SYS012. This is the same number as the one that appears in the ASSGN JCL statement for the file.

```
// ASSGN SYS012,X'nnn'
// TLBL SEQIN ...
```

This is how the JCL looks for the previous example.

Device Specifications

VSE only. This entry is optional for VSE Release 2.1 and above. It is required for releases prior to 2.1. You must identify the device on which the file resides or is to reside if the file is not VSAM. A VSAM file does not require a device specified. If omitted, the default is standard labeled tape with the rewind option in effect.

TAPE Label Rewind (Options)

The file resides, or is to reside, on tape. Following the keyword TAPE, you can code information regarding tape labels or rewind options:

Label:

T	The tape file has a leading tape mark or is to be written with a leading tape mark.
NL	The tape is, or is to be, unlabeled.
SL	Default if omitted. The tape has standard labels or is to be written with standard labels. Remember to include your TLBL statement in the JCL if it is standard labeled.

Rewind Options:

REWIND	Default if omitted. Rewind the tape before and after the run, but do not unload it.
UNLOAD	Rewind the tape before and after the run and unload it at the end.
NORWD	Do not rewind the tape before or after the run.

```
FILE MASTER INPUT FB 100 1000 SYS040 TAPE
```

A tape input file is on standard labeled tape with the option to rewind the tape before and after the run, but no unload.

```
FILE ACCTFLE INPUT FB 200 2200 TAPE NL UNLOAD SYS010
```

An unlabeled tape is read. The tape is rewound before and after the run, and then unloaded.

```
FILE NEWFILE OUTPUT FROM XYZ FB 150 1500 TAPE NORWD SYS010
```

A standard labeled tape is created. The tape is not rewound before or after the run.

Multiple File Input

This section describes how to process multiple files.

TAPE nn

If you have multiple tape files (not a multi-volume file) that you want to read in, the number (1 to 99) of files can be coded after the TAPE keyword. All of the files must have the same type of labels, format, record size, and block size.

```
FILE FILEIN INPUT FB 100 1000 TAPE 5 UNLOAD SYS010
```

Here, five separate standard labeled files are read in as one file. The rewind option specifies that the final tape is to be unloaded. All intermediate reels are unloaded as they are finished.

```
FILE FILE2 INPUT FB 200 2200 TAPE 6 NORWD SYS010
```

Above, six standard labeled files reside on one reel. If you code NORWD, all six files are read from the same reel. There is no rewinding and unloading between files.

TAPE NUMFILES or DISKETTE NUMFILES

If the number of multiple input files varies from run to run, you can dynamically specify the number in a VISION:Results program. Instead of coding a number after the TAPE keyword in the FILE statement, code:

```
... TAPE NUMFILES dataname ...
```

If you are using a diskette, code:

```
... DISKETTE NUMFILES dataname ...
```

You can code any valid data name. You do not have to define the field to which you are assigning the name. VISION:Results does this automatically. Then, in your program logic, sometime before the first file is completely processed, move the number of files you want to read to the field identified by the data name.

```
FILE SYSIPT CARD
  INNUM 2 1 NU
FILE TAPEIN INPUT FB 100 1000 TAPE UNLOAD NUMFILES INFILES
  SYS029
  ACCOUNT 3 4 PD
FILE TAPEOUT OUTPUT FB 100 2000 TAPE SYS030
  FROM TAPEIN
ON ONE MOVE INNUM TO INFILES ENDONE
SORT TAPEIN USING ACCOUNT WORK 3
```

```
WRITE TAPEOUT  
FIN  
05
```

Multiple tape inputs are read in, sorted in ACCOUNT sequence and written to one output tape file. The number of files varies from run to run, so the NUMFILE data name option is used in the FILE statement, and the number is obtained dynamically from card input and moved to the NUMFILES field.

The NUMFILES option comes in handy for programs that you are freezing.

Do not confuse multi-volume tape files with multi-files. If you have a file that spans several tape reels, you do not have to specify anything special. The VSE system treats it as one file provided that it has standard labels.

When reading multiple files, you cannot use the alternate drive assignment feature of VSE. All tapes must be read from the same drive.

DISK nnnn

The file resides, or is to reside, on a disk device. For VSE releases prior to r2.1, the supported values for nnnn are:

- 2311
- 2314
- 3330
- 3340
- 3350
- 3375
- 3380
- FBA (for use with 3310 and 3370 drives)

All models of the above devices are supported.

```
FILE ACTFILE INPUT FB 50 2000 DISK 3340
```

For VSE r2.1 and above, the disk device type entry is optional. If you make an entry, specify DISK or TAPE only, without a system number.

DISKETTE (options)

The file resides or is to reside on a 3540 diskette. After the keyword, specify one or more of the following options for data checking and positioning of the diskette:

NOFEED	Do not place the diskette into the collector's bin at end of file.
DATAVER	Check for indication on the diskette that the data on it has been verified.
VOLVER	Verify the volume serial number on the diskette.
VERIFY	Perform the two operations specified by DATAVER and VOLVER.
SECURE	On output, ensure that the data is secure.
PROTECT	On output, ensure that the file cannot be over-written.

```
FILE TRANSIN INPUT F 80 DISKETTE DATAVER SYS011
```

A diskette file is read with the option to check for the indication that the data on it has been verified.

ONEBUFF

VSE only. Code this if you want to allocate only one buffer to hold the data blocks of files read or written. This can conserve memory or create more wait time in the partition.

```
FILE SEQIN INPUT FB 200 2000 DISK 3340 SYS011 ONEBUFF
```

A sequential input file on disk is defined and one buffer is specified.

DYNAM

z/OS only. Indicates that the file (file name) is dynamically allocated by the VISION:Results program using the ALLOCATE command. Any file that specifies DYNAM must not have DD statements included for the file name in the JCL that will execute the VISION:Results program.

If DYNAM is coded for a file name, the program must explicitly issue a READ / WRITE / READDIR / READMEM / WRITEDIR / WRITEMEM command to perform the I/O, or else a validation error occurs. For details about the ALLOCATE command, see [Chapter 14: Using Procedural Commands](#).

```
FILE OUTFILE  OUTPUT  FB  100  FROM  OUTFILE  DYNAM
      OUTREC              100

WORKAREA
  JCLSTAT              1

WORKAREA
  OUTJCL              320      1
  FILLER              80      1
  VALUE  '//OUTFILE DD DSN=DYL.BCM.OUTFILE, '
  FILLER              80
  VALUE  '//          DISP=(NEW,CATLG,DELETE),UNIT=SYSDA, '
```

Figure 68 DYNAM (Page 1 of 2)

```

        FILLER      80
        VALUE  '//   SPACE=(TRK,(4,2),RLSE), '
        FILLER      80
        VALUE  '//   DCB=(RECFM=FB,BLKSIZE=2000,LRECL=100)

ON ONE
    ALLOCATE FILE DYNAMICALLY OUTJCL  STATUSFLAG JCLSTAT
    IF JCLSTAT EQ 'N'
        PRINT 'ERROR IN DYNAMICALLY ALLOCATED JCL'
    STOP
    ENDIF
ENDONE
WRITE OUTFILE

```

Figure 68 DYNAM (Page 2 of 2)

ISAM and VSAM Syntax Definitions

The following FILE statement specifications are for [ISAM](#) and VSAM files.

KEYLEN nnn

z/OS and VSE. Required if a VSE [ISAM](#) file is being defined. Required if [ISAM](#) is being defined and the processing is to be random retrieval, random retrieval with update, sequential I/O (VSAM), or skip sequential.

Required for VSAM KSDS RLS files. Not required for VSAM if either OPTION VSAMCAT or the DYLINSTL parameter VSAMCAT=Y was specified.

The keyword KEYLEN followed by a number from 1 to 255 specifies the length of the [ISAM](#) or VSAM KSDS key.

KEYLOC nnnn

z/OS and VSE. Required if a VSE [ISAM](#) file is being defined. Required for VSAM KSDS RLS files if the key position is not position 1 in the file. Not required for VSAM KSDS or z/OS [ISAM](#). The keyword KEYLOC followed by a number identifies the location of the key in the record (the first byte of the record is KEYLOC 1).

```

FILE ISFILE ISAM RANDOM SRCHKEY DISK 3330 FB 200 2000
  KEYLEN 7 KEYLOC 2 SYS008

```

In a VSE run, ISFILE is an [ISAM](#) file read randomly. It has a key length of 7 and the key starts with the second byte of the record.

STATUS dataname

z/OS and VSE. This is optional. It must be used during [ISAM](#) or VSAM random or skip processing. After the keyword STATUS, code any valid data name. You do not have to define the field anywhere. VISION:Results automatically allocates the field.

The field identified by the data name holds the status of a random or skip sequential retrieval. Returned codes are:

blank	Requested record not found.
I	Duplicate or invalid key.
Y	Requested record retrieved.
E	End of file reached (skip mode only).

After issuing a READ, check this field to make sure you are retrieving properly.

POSITION dataname

z/OS and VSE. Use this only if processing an [ISAM](#) or VSAM file in SKIP mode. This specifies whether processing is to start at or after the search key you supply. This keyword is optional.

After the POSITION keyword, code a valid data name to hold the positioning value. You do not have to define this field. VISION:Results automatically allocates the field.

During your program logic, move the desired value to the data name before issuing the first READ. The following values are supported:

blank	Position the file at a record equal to or greater than the partial or full key specified (default).
G	Same as above.
E	Position the file at a record equal to the partial or full key specified.
R	Repeat. You must supply this value each time you want to reposition at the same key that was used for the last positioning request.

```
FILE ISFILE ISAM SKIP SRCHKEY KEYLEN 7 KEYLOC 1
  POSITION SETIT STATUS ISFSTAT
  FB 200 2000 DISK 3330 SYS011
```

In a VSE run, an [ISAM](#) file is read in SKIP mode. Move a generic or full key to the SRCHKEY field and move a positioning code to SETIT before issuing a read.

PARTKEY nnn

z/OS and VSE VSAM KSDS. This is required if file is VSAM KSDS and processing is SKIP sequential read with positioning to start at or after a partial or generic key.

After the keyword PARTKEY, code the length (1 to 255) of the generic key to be searched for.

```
FILE VSFILE KSDS F 200 SKIP STARTKEY KEYLEN 7 PARTKEY 2
  POSITION SETIT
```

In a z/OS run, a VSAM KSDS file is read in SKIP mode. Only the first 2 bytes of the 7-byte key are used in the positioning. Before the first read is issued, a 2-byte generic key must be moved to STARTKEY and the positioning code moved to SETIT.

REUSE

z/OS and VSE. Optional. Required for VSAM ESDS or KSDS sequential output if you want to write over an existing VSAM file. The file you are applying the REUSE to must have been defined as reusable in the IDCAMS definition.

```
FILE VSFILE KSDS OUTPUT FROM XYZ F 200 REUSE
```

VSFILE is recreated without having been deleted in an IDCAMS run. The file has previously been established as reusable in an IDCAMS run.

ERASE dataname

z/OS and VSE. Code this if you intend to delete any records from a VSAM KSDS file during random I/O or sequential I/O processing.

After the keyword ERASE, code a valid data name. You do not have to define the field; VISION:Results does this automatically.

In your program, after you have read a record and now want to delete it from the file, move an E to the data name you have specified and then issue a WRITE. The record is removed from the file rather than rewritten.

```
FILE VSFILE KSDS IO RANDOM FINDKEY F 200  
KEYLEN 5 STATUS STATUSFLG ERASE DELETEFLG
```

A VSAM KSDS file is read randomly and updated. During program logic, after a record is retrieved, if it is determined that the record must be deleted, the value E must be moved to DELETEFLG before the WRITE is issued. This record is deleted rather than rewritten to the file.

RANDOM dataname (data type) and RANDOMX dataname

RANDOM—z/OS and VSE; RANDOMX—z/OS only. Code RANDOM when the file is [ISAM](#) or VSAM (KSDS/ESDS) and is either read randomly or read and updated randomly. Code RANDOMX only when the file is VSAM ESDS and is either read randomly or read and updated randomly.

The data name is required. You can code any valid data name to identify the field that holds the key of the record you are looking for. You do not have to define this field anywhere, because this area is automatically set up. You must move the appropriate key to this field before doing any random reads.

For [ISAM](#) or VSAM KSDS, if the key's data type is not character, code a data type of NU (numeric or zoned), PD (packed), or BI (binary) after the data name.

If you are retrieving records from an Extended Format VSAM ESDS file, the **RANDOMX** dataname identifies the field that receives your relative byte address (RBA) that is used instead of a key to retrieve and update ESDS records. The field identified by the data name is packed decimal and 16 bytes long.

If you are retrieving records from a VSAM ESDS file, the **RANDOM** dataname identifies the field that receives your relative byte address (RBA) that is used instead of a key to retrieve and update ESDS records. The field identified by the data name is binary and 4 bytes long.

For special considerations regarding signed and unsigned keys, see *Advantage VISION:Results for z/OS Getting Started*.

```
FILE ISFILE ISAM RANDOM FINDKEY ...  
.  
.  
.  
MOVE TRANSKEY TO FINDKEY  
READ ISFILE
```

Here, an [ISAM](#) file is being read randomly. It specifies that the key hold field is to be referenced by the name **FINDKEY**. During program logic, you need to move the key from wherever it resides to **FINDKEY** before you issue the **READ** command.

```
FILE MYVSAM KSDS IO RANDOM SEARCHKEY ...
```

Here, it indicates that the VSAM KSDS file is to be read and updated randomly. The data name **SEARCHKEY** is where the key is placed before reading and writing.

```
FILE VSAMIN ESDS INPUT RANDOM RBAKEY ...
```

A VSAM ESDS file is read randomly. The dataname **RBAKEY** identifies a 4-byte binary field set up by *VISION:Results* to hold the RBA of the record you want. Place the value into this area before issuing the **READ** command.

SKIP dataname (data type) and SKIPX dataname

SKIP—z/OS and VSE; **SKIPX**—z/OS only. Code **SKIP** when the input file is [ISAM](#) or VSAM (KSDS/ESDS) and you want to read it sequentially but do not want to start at the beginning. The **SKIP** mode allows you to begin at or above the full or generic key that you provide. Code **SKIPX** only when the input file is VSAM ESDS and you want to read it.

The data name is required if **SKIP** mode is specified. You can code any valid data name to identify the field that holds the full or generic key of the starting record. *VISION:Results* automatically sets up this area, so you do not have to define this field anywhere. Move the appropriate key to this field before doing any skip sequential reads.

For [ISAM](#) or VSAM KSDS, if the key's data type is not character, code a data type of **NU** (numeric or zoned), **PD** (packed), or **BI** (binary) after the data name.

If you are retrieving records from a VSAM ESDS file, the SKIP dataname identifies the field that receives your relative byte address (RBA) instead of a key to begin sequential reading in skip mode. You cannot use a generic RBA. You must specify a full RBA, and it must be equal to the RBA of the record with which you want to start. The field identified by the data name is binary and 4 bytes long.

If you are retrieving records from an Extended Format VSAM ESDS file, the SKIPX dataname identifies the field that receives your relative byte address (RBA) instead of a key to begin sequential reading in skip mode. You cannot use a generic RBA. You must specify a full RBA, and it must be equal to the RBA of the record with which you want to start. The field identified by the data name is packed decimal and 16 bytes long.

For special considerations regarding signed and unsigned keys, see *Advantage VISION:Results for z/OS Getting Started*.

```
FILE ISFILE ISAM INPUT SKIP STARTKEY ...
```

An [ISAM](#) file is read sequentially, but reading can be started and restarted at various points in the file during processing. In the example above, a full or generic (partial) key is moved to STARTKEY before issuing the first and subsequent reads.

```
FILE VSAMFIL KSDS F SKIP STARTKEY ...
```

In this example, a VSAM KSDS file is read sequentially, but reading can be started and restarted at various points in the file during processing. A full or generic (partial) key is moved to STARTKEY before issuing the first and subsequent reads.

```
FILE VSAMFIL ESDS F SKIP RBAKEY ...
```

Here, a VSAM ESDS file is read sequentially, but reading can be started and restarted at various points in the file during processing. An RBA that is equal to the RBA of the record at which you want to begin reading is moved to RBAKEY.

RELBYTE dataname and RELBYTEX dataname

RELBYTE—z/OS and VSE; RELBYTEX—z/OS only. This is optional. If you need to know the Relative Byte Address of the VSAM ESDS record just read or written, this option allows you to obtain that information.

After the keyword RELBYTE or RELBYTEX, code a valid data name to identify the field you can interrogate during processing. You do not have to define the field. It is done automatically. RELBYTE defines a 4-byte binary field; RELBYTEX defines a 16-byte packed decimal field for an Extended Format VSAM ESDS.

```
FILE VSAMIN ESDS F 200
  RELBYTE RBAIN
MOVE RBAIN TO DYLPRTCOMM
HEXPRINT VSAMIN
```

A VSAM ESDS file is read; each record's RBA is obtained from the RBAIN field and placed in the comments area of the file print line before the record is printed. The record is printed in hex and graphics, with its RBA appearing to the left.

PASSWORD 'password'

z/OS and VSE. VSAM only. Required if the VSAM file is password-protected. Following the keyword PASSWORD, code the necessary 1- to 8-character password enclosed in quotation marks.

```
FILE VSAMIN KSDS PASSWORD 'LETMEIN' F 140
```

A VSAM KSDS input file is defined and the password is given in the FILE statement.

PASSWORD dataname

z/OS and VSE. VSAM only. An alternate and dynamic means of supplying a password to access a protected file. Following the keyword PASSWORD, code any valid data name. You do not have to define the field. VISION:Results automatically allocates it for you.

In your program logic, before the file is opened and read, move the password to the field you specified in the FILE statement.

```
FILE VSAMIN KSDS PASSWORD UNLOCK F 140
```

```
ON ONE  
MOVE 'LETMEIN' TO UNLOCK  
ENDONE
```

```
READ VSAMIN  
PRINT VSAMIN  
FIN
```

A VSAM KSDS file is read and printed. Before the file is accessed, the password is set.

You can also read the password in from instream data or introduce it using the PARM parameter in the EXEC statement (z/OS only).

CYLOFL nn

VSE [ISAM](#) only. This is optional. It is required if the number of cylinder overflow tracks on each cylinder is not 0. Following the keyword CYLOFL, code the number of cylinder overflow tracks per cylinder allocated to the file.

```
FILE SEQIN FB 200 4000 TAPE SYS011  
FILE ISFILE ISAM OUTPUT FROM SEQIN FB 200 2000  
DISK 3330 KEYLEN 5 KEYLOC 2  
CYLOFL 2
```

A VSE [ISAM](#) fixed blocked file is created from sequential input. Two tracks per cylinder on the 3330 are established as cylinder overflow tracks.

EXTENTS nn

VSE [ISAM](#) only. Optional. Specifies the total number of cylinder index, prime data, and independent overflow EXTENTS coded in the JCL for the file. This number does not include the master index EXTENT statement, if present. Following the keyword EXTENTS, code the number as described above. If omitted, it defaults to three extents.

MASTER

VSE [ISAM](#) only. Code this keyword when defining an [ISAM](#) file and the file has a master index.

RETAIN

z/OS and VSE. RETAIN can be specified for any file except those that are being processed with MODIFY or EXIT. Use to indicate that the file description and all its associated data descriptions are to be saved and made available for use in a subsequent request.

If RETAIN is coded on a FILE statement, you can code USE and the file name in a subsequent report. This allows you to process the file again, but without completely defining it again.

If RETAIN has not been specified, an attempt to code USE and the file name in a subsequent report request results in a validation error.

EXIT and MODIFY

The following FILE operands are used for special file processing only.

MODIFY modulename [nK]

z/OS and VSE. The MODIFY keyword is used to indicate that special processing is required either immediately following the input operation or immediately prior to the output operation. The module name coded immediately following the keyword MODIFY is the name of a user written module invoked to perform the special processing.

For VSE users, this name must be followed by the memory requirement for the module, for example, 1K. The default is 4K. The maximum size allowed for VSE is 999K.

For z/OS users, the memory size is not required and there is no maximum module size. For rules concerning user modules, see [Chapter 14: Using Procedural Commands](#).

When MODIFY is specified, VISION:Results performs all the I/O operations against the file either as part of the automatic cycle or when READ/WRITE is executed. The user module is invoked to process the record either immediately following input or just prior to output.

Use this feature to automatically scramble records prior to writing out a file, to expand/condense records as they are processed, to secure fields within a record, or for many other applications.

Any other FILE operands that apply are still required.

```
FILE PAYROLL INPUT FB 360 7200 TAPE SYS011  
  MODIFY DECRYPT
```

A payroll file is read. When the file was created, it was encrypted for security reasons. Whenever a record is read, it is passed to a routine DECRYPT that decrypts it so that the records can easily be processed in this program.

When specifying data names following a FILE statement that has been coded with MODIFY, define the fields on input, as they appear after the routine has executed. For output files, define the fields as they appear prior to the routine being executed.

Restrictions

- You cannot output a file from a file that is specified with MODIFY. Specify the output file from (FROM) itself and then move the data yourself.
- You cannot specify a file print using the OPTION statement for a file that has MODIFY specified.
- To define an output file with MODIFY specified, code OUTPUT from itself.

EXIT modulename [nK]

z/OS and VSE. The EXIT keyword specifies a user module invoked to do all I/O processing for this particular file. This facility can access files that are not normally supported by VISION:Results. It is your responsibility to provide the module. For VSE users, the module name must be followed by the memory requirements for the module, for example, 1K. The default is 4K. The maximum allowable size for VSE is 999K. For z/OS users, the memory size is not required.

If EXIT is coded on a FILE statement, your module is invoked whenever I/O is required for this file, either as part of the automatic cycle or whenever READ/WRITE is executed.

Whenever a FILE specifying EXIT is coded, you may be required to supply a STOP statement. If there is a SORT in the program, this determines when the program terminates. If there is no SORT, you must execute a STOP when you have determined that the program is complete. For rules concerning user-written modules, see [Chapter 14: Using Procedural Commands](#).

```
FILE BDAMSTR FB 435 8700 INPUT EXIT USERIO
```

A file named BDAMSTR is going to be read by the user's module USERIO.

Restrictions

When coding a FILE statement that specifies EXIT, you must supply all the required operands to satisfy VISION:Results validation, even though they may not be required for the user module. Certain FILE operands are not valid if EXIT is specified. The following keywords are not valid with EXIT:

- LENGTH
- COUNT
- SELECT
- INTSEL
- NUMFILES
- POSITION
- SKIP/SKIPX
- RANDOM/RANDOMX
- ERASE
- PASSWORD
- PARTKEY

The STATUS keyword can be used with EXIT, but it is your responsibility to set the field to E in the module.

You can SORT a file that is being processed using EXIT, but you cannot specify an EXIT-processed file as the operand of UNTIL on a SORT statement.

You can process the EXIT file in the same manner as any other VISION:Results file, but you cannot specify file printing using the OPTION statement for this file.

You can define data names following your FILE statement for your EXIT file in the same manner as for a regular file. VISION:Results allocates a record area for your file equal to the larger of the record size or SIZE parameter, and the data names apply to this area. It is your responsibility to ensure that the record is moved into this area by the I/O module.

You can create an output file using an EXIT keyword, but there are some restrictions. An output EXIT file must be OUTPUT FROM itself. To write a record directly from another file, you must move it yourself. Also, if you want to OUTPUT another FILE from an EXIT file, you must move the record yourself. You cannot code FROM an EXIT file.

```
FILE SYSIN CARD
FILE RECFILE FB 80 8000 OUTPUT
  FROM RECFILE EXIT USERIO
MOVE SYSIN TO RECFILE LENGTH 80
FIN
```

This example writes data to RECFILE. The records are copied to the RECFILE.

SIZE nnnnn

z/OS and VSE. SIZE is valid only if EXIT or MODIFY has been supplied. It is used to indicate how large an area is required to hold a record.

MODIFY

When SIZE is used with MODIFY, it indicates that the record is to be processed in a work area whose size is specified by the 1- to 5-digit number following the keyword SIZE. All data definitions apply to this work area. If SIZE is specified, the user-written MODIFY module moves from the I/O area to this work area on input and from this work area to the I/O area on output. When SIZE is specified, it is not possible to access the actual I/O area in VISION:Results.

You can specify SIZE if your records are being expanded. For example, you are reading VB records that are being expanded in a MODIFY module to fixed length for easier processing in VISION:Results.

```
FILE VAREMP VB 210 2144 MODIFY USEREXP
  SIZE 400
```

A variable record file is read by VISION:Results. Immediately following the input operation, the user's module USEREXP is invoked to expand the variable record to a fixed 400-byte record. All data names specified following this statement refer to the expanded data in the 400-byte area.

```
FILE VAREMP VB 210 2144 MODIFY USERCND
  SIZE 400 OUTPUT FROM VAREMP
```

You build a record in a 400-byte area. Just prior to the output operation, the module USERCND is invoked to condense the record in the 400-byte area down to a variable record, maximum 210 bytes in the I/O area.

EXIT

When coding an EXIT-processed file, indicate the maximum record length possible. This is necessary to allow VISION:Results to allocate a record area. You can specify the record size either as the logical record length following the record format keyword or by specifying SIZE.

If both SIZE and a logical record length are supplied, VISION:Results allocates an area equal to the size of the larger. If neither is supplied, a validation error occurs. It is the responsibility of the user-written EXIT module to move the record to this area on INPUT and write from this area on OUTPUT.

PARM (dataname)

z/OS and VSE. The PARM keyword is valid only if EXIT or MODIFY has been specified. The keyword PARM is followed by a series of data names enclosed in parentheses, indicating the parameters that are to be passed to the EXIT/MODIFY module. The data names cannot be more than 10 characters. A standard parameter list is built using the specified data names and is pointed to by register 1. Standard linkage conventions apply. EXIT and MODIFY both have default parameter lists. The PARM keyword needs to be supplied only if the user module requires a parameter list that differs from the standard. For a description of writing user EXIT and MODIFY routines and the default parameter list, see [Chapter 14: Using Procedural Commands](#).

Two special keywords can be specified as part of the parameter list—IOAREA and WORK. They are valid only as part of the operand field of the PARM keyword.

IOAREA

When specifying a parameter list for a MODIFY routine, you must be able to specify the I/O area for this particular file. Specify the keyword IOAREA as part of the parameter list. The address of the I/O area for the current record is passed as the appropriate entry in the list.

IOAREA refers to the area that VISION:Results reads into or writes from. If SIZE is also specified, it is your responsibility, in the MODIFY routine, to move the record from this area to the work area on input or to this area from the work area on output.

IOAREA is invalid for EXIT.

WORK

WORK is valid for EXIT-processed files and any MODIFY file that has SIZE specified. It refers to the area that has been allocated by VISION:Results to hold the record while it is being processed. It is your responsibility to move to it during an INPUT EXIT/MODIFY module and from it during an OUTPUT EXIT/MODIFY module.

IOU-Processed Files

Any program that requires more than eight (seven if SORT is being used) input files or eight output files can process the additional files using the IOU (Input Output Unlimited) facility. The IOU facility calculates how many IOU files it needs and allocates them appropriately. This facility might also be required if you get a '628E Unable to allocate work area' message.

The IOU facility can process any standard sequential file (fixed, variable, blocked, unblocked, input, output) on tape or disk. Spanned files are not supported, and neither is VSAM or [ISAM](#). There are some limitations associated with IOU files:

- For variable files, VISION:Results does not manipulate the RDW. It is your responsibility to allow for it.
- Certain FILE keywords are not valid with IOU-processed files. See EXIT restrictions for the list of keywords you cannot use.
- Any program using the IOU facility that does not contain a SORT must determine when processing is to terminate and issue a STOP.

To use the IOU facility, code your FILE statement for your sequential file specifying all the required keywords. You must also specify EXIT DYLIU on the statement. This causes the module DYLIU to perform all I/O for the file. Do not code a PARM keyword. Always specify a maximum record size.

```
FILE SEQTape VB 85 894 SYS011 TAPE  
EXIT DYLIU
```

For VISION:Results logic purposes, this is a standard file; however, the I/O is performed using the module DYLIU.

If you get a 628E message, making some of your output files IOU-processed files might allow the program to compile and execute.

Considerations for Processing Variable-Length Records

VISION:Results simplifies the processing of variable files. You do not have to worry about differences in processing VSAM and non-VSAM variable files because both are handled the same way.

Record Size

When specifying the logical record length for a variable file, you must specify the maximum data length. For non-VSAM files, VISION:Results adjusts this length to allow for the RDW (record descriptor word) unless exit DYLIU is used.

Block Size

Blocking has no meaning for VSAM files and is ignored. When specifying the maximum block size for non-VSAM files, you must specify the actual physical size of the block. In calculating this, add 4 to the specified record length (to allow for the RDW unless exit DYLIOW is used), multiply by the blocking factor and then add 4 to allow for the BDW block descriptor word.

For example, a variable file containing records with a maximum length of 200, blocked 10, has a maximum block size of 2044. $((200+4)*10+4)$. The maximum block size specified for a variable file does not have to allow for a fixed number of the maximum length records. The specific application determines what the optimum block size must be. VISION:Results, however, requires that the specified block size be at least 8 greater than the record size given.

Location of Data Within Record

VISION:Results assumes that position one of every variable record is the first byte of data. It automatically takes care of the 4-byte RDW for non-VSAM variable files. If you are setting up data definitions for variable-length records, start defining fields from the first byte of data. Exception: If exit DYLIOW is used, you must account for the RDW and the data starts in byte 5.

Length of Current Record

On input, VISION:Results keeps track of the length of the data in the record just read. You can access this information by specifying the keyword LENGTH and a data name. The data name specified is a 2-byte binary field and contains the length of the data in the record just read. This applies to both VSAM and non-VSAM files.

For output files, it is your responsibility to tell VISION:Results how long the data portion of the current record is. For non-VSAM files, VISION:Results uses this length to build a RDW before writing the record.

Output from Itself

If a file is defined to be output from itself, it has its own I/O area assigned. It also has a 2-byte binary length field assigned to the file. You can reference this length field by specifying the keyword LENGTH followed by a data name.

It is your responsibility to move data to the area and place the length of the data in the LENGTH dataname statement before the record is written using either the automatic cycle or a WRITE command. In most cases, failure to move a valid length to the LENGTH dataname statement results in wrong length records being written.

Output from an Input File

When a file is written from an input file, VISION:Results does not assign an I/O area for the file or a length field. VISION:Results references the I/O area and length field for the input file when the output file is written. VISION:Results places restrictions on the type of file that a variable output file can be written from.

A variable non-VSAM file can be written only from another variable non-VSAM file. A variable VSAM file cannot be written from a variable non-VSAM file. If other combinations are required, you must specify OUTPUT from itself for the file, move the data, and set up the data length.

When a record is being written from an input area, the length automatically is set up by VISION:Results when the record is read. LENGTH cannot be specified for the output file because the length field does not exist for this file. You can specify the LENGTH dataname for the input file if you want to adjust the length of the record. You can shorten a record to be written by changing the contents of the LENGTH dataname. Do not attempt to write a longer record.

If it is necessary to increase the length of a record, specify OUTPUT from itself on the FILE statement, move the data, and set up the length. The LENGTH data name always contains the length of the data only.

File Printing Variable Files

VISION:Results automatically handles the RDW; you need only be concerned with data. There is one exception. Whenever VISION:Results does a file print of the record using either the OPTION statement or one of the file print commands such as HEXPRINT or PRINT, VISION:Results prints the RDW if it is present. An RDW is present only for non-VSAM files. This is done to assist in debugging. The data that is defined at byte 1 in the data definitions is at byte 5 in the file print of the record if there is an RDW. This is also true for the dump of the record in a PRINTERERROR error analysis dump.

Sorting a Variable File

To eliminate a possible error during SORT, VISION:Results moves variable-length records from the I/O area to a fixed-length area for processing. This is transparent and you normally do not need to worry about it. If an error occurs in the VISION:Results execution, the record shown in the I/O area in the PRINTERERROR dump is not necessarily the record you are processing. For both VSAM and non-VSAM files, the variable record has been moved to a work area. Check the record there.

When you use SORT, you can have a maximum of seven input files.

Exceptions to the Above Rules

VISION:Results assumes that you are working only with the data portion of the record and that it handles the RDW if it exists. There are two exceptions to this:

- If the file is an EXIT-processed file defined as variable, it is your responsibility to handle any RDW that there might be. VISION:Results does not add 4 bytes to the record size to allow for an RDW, nor does it assume that the first byte you are defining is data.
- Under z/OS, it is not necessary to specify the record format. If you do not specify the record format, VISION:Results does not make allowances for a possible RDW.

Always specify at least the record format for files. The following example can result in creating an output file that has two RDWs on every record.

z/OS

```
FILE INFILE LENGTH INLEN          1
FILE OUTFILE VB 500 5044 LENGTH OUT 2
LEN
OUTPUT FROM OUTFILE              3
MOVE INFILE TO OUTFILE LENGTH INLEN 4
MOVE INLEN TO OUTLEN             5
```

- Statement 1 defines an input file but does not specify the record format. INFILE refers to byte 1 of the physical record, not byte 1 of the data. If INFILE is in fact a variable-length file, INLEN contains the length of the record including the 4-byte RDW and INFILE refers to the first byte of the RDW.
- Statement 2 defines an output file. It is to be a variable file and VISION:Results automatically allows for the RDW.
- OUTFILE refers to byte 1 of the data or byte 5 of the physical record.
- Statement 4 moves the data from the input area to the output area. The RDW from the input record is at byte 5 of the physical output record.
- Statement 5 sets up the data length for the output file. When VISION:Results enters the automatic cycle to write the output file, it adds 4 to the output length to allow for the RDW and then builds the RDW at byte 1 of the physical output record. When the record is written, it has two RDWs with the actual data starting at location 9.

Avoid this problem by specifying the record format in the file statement. In this case:

```
FILE INFILE VB LENGTH INLEN
```

Considerations for Unblocked ISAM Files

Note: This section only applies to VSE.

Handling [ISAM](#) files by the IBM VSE I/O routines is different for blocked than it is for unblocked. The following sections outline the differences and provide guidelines for processing unblocked [ISAM](#) files.

Sequential Read

When a blocked [ISAM](#) file is read, the system brings in only the data record, not the key prefix. When an unblocked [ISAM](#) file is read sequentially, the key precedes the data in the record area even though the key may also be embedded in the record.

Read an unblocked [ISAM](#) file whose record length is 35 and key length is 5.

```
FILE ISAMIN F 35 ISAM DISK 3340 KEYLEN 5 KEYLOC 1
FIRSTBYTE 1 6
NEXTFLD 10 7
```

The first byte of data is located in byte 6 of the record area. The last 5 bytes of data are beyond the end of the record area, strictly speaking. If you need to access the last n bytes of the record (where n is the key length), set up a WORKAREA and a variable-length field and move into it to obtain the entire record:

```
MOVE FIRSTBYTE LENGTH 35 TO HOLDREC
```

Sequential Load/Extend

Unblocked [ISAM](#) sequential output requires that the record be written from an area containing the key followed by the data.

Create an unblocked [ISAM](#) file with a record length of 35 and a key length of 5.

```
FILE ISAMOUT F 35 ISAM DISK 3340
KEYLEN 5 KEYLOC 1 OUTPUT FROM WORKFL
FILE WORKFL F 40 TAPE NL SYS029
WORKKEY 5 WORKREC 35
```

The file WORKFL is not a real file, just an area to write from. In this example, SYS029 must be assigned to IGN (ignore) in the JCL. The key of the output record is moved to WORKKEY and the data to WORKREC.

Random Retrieval

Random retrieval of unblocked [ISAM](#) files works the same way as for blocked [ISAM](#) files. Just move the key you are searching for to your RANDOM data name and issue a read. The data is returned in your record area WITHOUT the key prefix. Byte 1 of the data is in byte 1 of the record area.

Random Update

In this type of access, the record must be randomly retrieved first. See above.

If the record is found (STATUS dataname = Y), only the replacement record is moved to the output area. If the record is not found, the key *and* data are moved to the output area. In both cases, a WRITE command is then issued to replace or add the record as necessary.

```
FILE ISAMUPD F 35 ISAM DISK 3340 RANDOM IKEY
  KEYLEN 5 KEYLOC 1 IO FROM WORKFL
  RECIN 35
FILE WORKFL F 40 TAPE NL SYS029
  OLDREC 35 1
  NEWKEY 5 1
  NEWREC 35 6
```

The file WORKFL is not a real file, just an area that you can write in. You must assign SYS029 to IGN.

Considerations for Concatenated Input Files

Note: This section applies to z/OS only.

VISION:Results supports concatenation of unlike input data sets only for data sets on different device types or with different block sizes. All data sets in the concatenation list must have the same LRECL (record length) and RECFM (record format).

Chapter 7: Using the IF Command

Use the IF statement to selectively process data, such as when only records of a certain type on a file are wanted or when different logic is to be executed depending on conditions.

Comparing and Selecting Data

You can code IF statements almost anywhere in your program, as part of regular logic or in special logic, such as ON END, ON CHANGE, or ON ONE.

For example, in a simple file print program, you want to select records with a state code of CA. To accomplish this, code:

```
FILE FILEIN INPUT FB 200 2000
  STATECODE 2 45
IF STATECODE NE 'CA' REJECT ENDIF
PRINT FILEIN
```

If the state code above is not equal to CA, the record is rejected. Otherwise, execute the next statement following the ENDIF (PRINT FILEIN). The IF statement can also be stated:

```
IF STATECODE EQ 'CA' NEXT ELSE REJECT ENDIF
PRINT FILEIN
```

If the state code is equal to CA, execute the next statement following the ENDIF (PRINT FILEIN); otherwise, reject the record.

You can also code:

```
IF STATECODE EQ 'CA' PRINT FILEIN
  ELSE REJECT ENDIF
```

The format of an IF (conditional) statement is:

```
IF datanamex [NOT] relationaloperator {datanamez}
  true condition-1 imperative statement(s)
ELSE | OTHERWISE {literal}
  false condition imperative statement(s)
ENDIF
```

Figure 69 IF (conditional) Statement

IF

IF signals the beginning of a conditional statement.

datanamex This represents a field, identified by its data name, that is to be compared to another data name or to a literal, or is to be tested for a certain state, such as positive, negative, numeric, or bits on.

This field can be alphanumeric, packed, or binary and can have from zero to nine decimal positions. It can be indexed, and its length cannot be modified by the LENGTH xx phrase.

NOT

NOT is an optional keyword that negates the relational operator with which it is associated. For example, if condition-1 is true, then NOT condition-1 is false. Likewise, if condition-1 is false, then NOT condition-1 is true.

Condition	Truth	Value
condition-1	True	False
NOT condition-1	False	True

Relational Operators

Relational operators represent the type of comparison to be executed. Valid entries are:

- EQ** Equal. If datanamex is equal to datanamez or literal, execute true condition imperatives. If not equal, execute false condition imperatives.
- NE** Not Equal. If datanamex is not equal to datanamez or literal, execute true condition imperatives. If equal, execute false condition imperatives.
- LT** Less Than. If datanamex is less than datanamez or literal, execute true condition imperatives. If datanamex is equal to or greater than datanamez or literal, execute false condition imperatives.
- GT** Greater Than. If datanamex is greater than datanamez or literal, execute true condition imperatives. If datanamex is less than or equal to datanamez or literal, execute false condition imperatives.
- LE** Less Than or Equal To. If datanamex is less than or equal to datanamez or literal, execute true condition imperatives. If datanamex is greater than datanamez or literal, execute false condition imperatives.
- GE** Greater Than or Equal To. If datanamex is greater than or equal to datanamez or literal, execute true condition imperatives. If datanamex is less than datanamez or literal, execute false condition imperatives.

Dataname or Literal

This represents a field, identified by its data name, or a literal that is being compared to. This field can be alphanumeric, packed, or binary and can have zero to nine decimal positions.

The data type or number of decimal positions does not have to be the same as datanamex. Data conversion and decimal alignment are handled automatically before the compare is made. The field can be indexed, and its length cannot be modified by the LENGTH xx phrase.

VISION:Results also allows a virtually unlimited number of literals as comparators within IF statements. For example:

```
IF FIELD1 EQ 'LITERAL1' 'LITERAL2' 'LITERAL3'
  NEXT
ELSE
  REJECT
ENDIF
```

Figure 70 Literals as Comparators

True Condition-1 Imperatives

If a condition is true depending on the test made, you can execute almost any VISION:Results command. This includes MOVE, CALL, LIST, READ, WRITE, and arithmetic statements. However, you cannot code a SORT or another IF statement. Nested IFs are supported in STRUCTURED mode.

You can branch to other logic (GOTO), ACCEPT, REJECT, terminate the run (STOP), or move on to the next statement following ENDIF (NEXT).

```
IF COSTCENTER EQ 5443
  MOVE 'X' TO FIELD A
  COUNTER = COUNTER+1
  WRITE FILEOUT
  ACCEPT
ELSE REJECT ENDIF
```

If the COSTCENTER value is equal to 5443, the imperative move, add, and write statements are executed and the program logic is exited using ACCEPT. If the COSTCENTER value is not equal to 5443, the logic is exited using REJECT.

If, at the end of your true condition imperatives, you do not ACCEPT, REJECT, GOTO, or STOP, execution then proceeds to the next statement following ENDIF.

```
IF COSTCENTER EQ 5443
  MOVE 'X' TO FIELD A
  COUNTER = COUNTER+1
  WRITE FILEOUT
ELSE REJECT ENDIF
```

This example is the same as the preceding one, except that the true condition logic is not exited using ACCEPT. The next statement executed is whatever is present after the keyword ENDIF.

False Condition Imperatives

If a condition is false depending on the test made, you can execute almost any VISION:Results command. This includes MOVE, CALL, LIST, READ, WRITE, and arithmetics.

You cannot code a SORT or another IF statement. You can branch to other logic (GOTO), ACCEPT, REJECT, terminate the run (STOP), or fall through to the next statement.

```
IF COSTCENTER NE 5443 REJECT
ELSE
  MOVE 'X' TO FIELDA
  COUNTER = COUNTER+1
  WRITE FILEOUT
  ACCEPT ENDIF
```

If the COSTCENTER value is not equal to 5443, the program logic is exited using REJECT. Otherwise, the move, add, and write imperatives are executed and the logic is exited using an ACCEPT.

The word ELSE precedes the false condition imperatives to indicate their beginning.

You do not have to code false condition imperatives. The preceding example can also be coded as follows:

```
IF COSTCENTER NE 5443 REJECT ENDIF
MOVE 'X' TO FIELDA
COUNTER = COUNTER+1
WRITE FILEOUT
ACCEPT
```

If you do not code false condition imperatives as part of the IF statement, the statements following the ENDIF terminator are executed. If no executable statements (such as MOVE or LIST) follow ENDIF, or if the following statements are part of special logic (such as ON CHANGE or ON END), an automatic ACCEPT happens and your program logic is exited.

ENDIF

ENDIF terminates the IF statement and is always required. Code this following the imperative statements associated with the IF.

For a list of simple valid compare combinations and the rules that govern them, see [Chapter 3: Syntax Rules](#).

Range Compares

You can incorporate a compare for a range or ranges of values in your IF statement using the relational operators EQ or NE:

```
IF DEPT EQ 910 THRU 915 ACCEPT ELSE REJECT ENDIF
```

Here, if DEPT is equal to 910, 911, 912, 913, 914, or 915 exit the program logic using ACCEPT. Otherwise, exit the logic using REJECT.

```
IF DEPT EQ 910 THRU 915 935 THRU 940 ACCEPT
  ELSE REJECT ENDIF
```

In this example, more than one range compare has been expressed in a statement.

Series Compares

You can incorporate a compare for a series of values in your IF statement using the relational operators EQ or NE:

```
IF DEPT EQ 910 911 913 ACCEPT ELSE REJECT ENDIF
```

If DEPT is equal to 910, 911, or 913, exit the program logic using ACCEPT. Otherwise, exit the logic using REJECT.

Combined Range and Series Compares

You can combine the range and series tests into one IF statement.

```
IF DEPT EQ 401 517 910 THRU 915 934 937 950 THRU 954
  ACCEPT ELSE REJECT ENDIF
```

If DEPT is equal to 401, 517, 910 through 915, 934, 937, or 950 through 954, exit the program logic using ACCEPT. Otherwise, exit the logic using REJECT.

Class or Status Tests

There are occasions when, in order to select certain data or execute certain logic, you need to know if a field is in a particular status. You want to determine if it is positive or negative, contains numeric data only, or if certain bits are on. Different versions of the IF statement accomplish these things for you.

Test for Positive Value

You can test a numeric (NU), packed (PD), or binary (BI) field for a positive (greater than 0) value using this format:

```
IF datanamex [NOT] POSITIVE
  true condition imperative statement(s)
ELSE or OTHERWISE
  false condition imperative statement(s)
ENDIF
```

Figure 71 Test for Positive Value

The data name can be indexed. It cannot have its length modified by the LENGTH xx phrase.

```
IF BALANCE POSITIVE
  PRINT ARREC
  ACCEPT
ELSE REJECT ENDIF
```

If the BALANCE field is greater than 0, execute the true condition imperatives (PRINT and exit program logic using ACCEPT). Otherwise, exit the logic using REJECT.

Test for Negative Value

You can test a numeric (NU) field, packed (PD) field, or binary (BI) field for a negative (less than 0) value using this format:

```
IF datanamex [NOT] NEGATIVE
  true condition imperative statement(s)
ELSE or OTHERWISE
  false condition imperative statement(s)
ENDIF
```

The data name can be indexed. It cannot have its length modified by the LENGTH xx phrase.

```
IF BALANCE NEGATIVE
  PRINT ARREC
  ACCEPT
ELSE REJECT ENDIF
```

If the BALANCE field is less than 0, execute the true condition imperatives (PRINT, and exit program logic using ACCEPT). Otherwise, exit the program logic using REJECT.

Test for Numeric Value

You can test a 1- to 31-byte character (CH), numeric (NU), or a 1- to 16-byte PD field for a numeric value, using this format:

```
IF datanamex [NOT] NUMERIC
  true condition imperative statement(s)
ELSE or OTHERWISE
  false condition imperative statement(s)
ENDIF
```

Figure 72 Test for Numeric Value

For NU field	The result is true only if the contents of the field contain external decimal numbers 0 to 9 (hex notation F0 to F9) with the exception of the low order or least significant byte, which can be A0 to A9, C0 to C9, E0 to E9 (signed positive), or B0 to B9, or D0 to D9 (signed negative).

For PD field The result is true only if the contents of the field contain external decimal numbers 0 to 9 (hex notation F0 to F9) with the exception of the last (sign) position, which is tested for a value of A through F.

For NU or CH fields, the IF NUMERIC test can be restricted to allow the sign position to contain F0-F9 only, by use of the DYLINSTL parameter NUMCHAR=Y, or the statement OPTION NUMCHAR. For PD fields, the IF NUMERIC test can be restricted to allow the sign position to contain a value of C only.

The datanamex field can be indexed. It cannot have its length modified by the LENGTH xx phrase.

For example, for a field called NUMFIELD that is defined as a 5-byte numeric field, the following values are considered numeric:

12345	(F1F2F3F4F5)
1234E	(F1F2F3F4C5)

The following values are considered non-numeric:

1 ~~b~~ 1345	(F140F1F3F4F5)
~~b~~ 1345	(4040F3F4F5)
123 ~~b~~ ~~b~~	(F1F2F34040)

Test for Bits On

You can test a 1-byte field of any data type for bits on (ones) using the following format:

```
IF datanamex [NOT] ONES hex-literal-mask
  true condition imperative statement(s)
ELSE or OTHERWISE
  false condition imperative statement(s)
ENDIF
```

Figure 73 Test for Bits On

The hex-literal-mask is used as an 8-bit mask. The bits of the mask correspond one for one with the bits of the field represented by datanamex. A mask bit of 1 indicates that the corresponding bit in the field is to be tested. When the mask bit is 0, the corresponding bit in the field is ignored. When specifying a test for bits ONES, for a true condition to be set, all of the bits in the field must be ones that are also ones in the mask literal.

The datanamex field can be indexed. It cannot have its length modified by the LENGTH xx phrase.

```
IF FIELD1 ONES X'C8'
  GOTO PROCA
OTHERWISE
  GOTO PROCB ENDIF
```

Bits 0, 1, and 4 in FIELD1 are tested for ones. If all the tested bits are on (1), there is a branch to PROCA. If none or only one or two of the bits are on, the logic identified by the PROCB tag is entered.

Test for Bits Mixed

You can test a 1-byte field of any data type for bits mixed using the following format:

```
IF datanamex [NOT] MIXED hex-literal-mask
    true condition imperative statement(s)
ELSE or OTHERWISE
    false condition imperative statement(s)
ENDIF
```

Figure 74 Test for Bits Mixed

The hex-literal-mask is used as an 8-bit mask. The bits of the mask are made to correspond one for one with the bits of the field represented by datanamex. A mask bit of 1 indicates that the corresponding bit in the field is to be tested. When the mask bit is 0, the corresponding bit in the field is ignored. When specifying a test for bits MIXED, for a true condition to be set, at least one of the bits in the field must be 1 that is also 1 in the mask. If all of the tested bits in the field are ones, a true condition is still set.

The datanamex field can be indexed. It cannot have its length modified by the LENGTH xx phrase.

```
IF FIELD1 MIXED X'F0'
    GOTO PROCA
ELSE
    GOTO PROCB ENDIF
```

Bits 0, 1, 2, and 3 in FIELD1 are tested for at least one bit on (1). If one or all bits are on, there is a branch to PROCA. If none of the tested bits is on, there is a branch to PROCB.

Compound Compares

If you need to compare for several conditions before a decision to select or process can be made, you do not have to do each compare separately. You can code compound conditions in your IF statements.

Select or Process if Any of Several Conditions Is True

An OR connector between two or more compares indicates that if any of the compares results in a true condition, the true condition imperative statements are executed. The format is as follows:

```
IF datanameu [NOT] relational operator {datanamew}
                                     {literal }
OR
  datanamex [NOT] relational operator {datanamew}
                                     {literal }
  true condition imperative statement(s)
ELSE or OTHERWISE
  false condition imperative statement(s)
ENDIF
```

Figure 75 OR Connector Format

```
IF DEPT EQ 201 OR
  JOBCLASS EQ 'C'
  ACCEPT
ELSE REJECT ENDIF
```

A record is accepted for output if the value of DEPT is equal to 201 or if JOBCLASS is equal to C. If neither is true, the record is rejected.

```
IF FIELDA EQ FIELDC OR
  FIELDH (INW) GT FIELDJ OR
  FIELDM LT 721
  COUNTER=COUNTER+1
  GOTO PROCA
ELSE GOTO PROCB ENDIF
```

If any of the compares is true, the COUNTER field is incremented by 1 and a branch to PROCA occurs. If none of the compares is true, a branch to PROCB occurs.

You can include series, range, or special tests (POSITIVE, NEGATIVE, NUMERIC) in your compound OR compares:

```
IF FIELDA NEGATIVE OR
  FIELDDB EQ 401 403 511 THRU 513
NEXT ELSE REJECT ENDIF
```

Figure 76 Compound OR Compare

If FIELDA contains a negative value or if FIELDDB contains the value 401, 403, 511, 512, or 513, execute the statement following ENDIF. Otherwise, if none of the compares is true, reject the record.

Select or Process if All of Several Conditions Are True

An AND connector between two or more compares indicates that only if all of the compares result in a true condition will the true condition imperative statements be executed. The format is as follows:

```
IF datanameu [NOT] relational operator {datanamew}
    {literal }
AND
    datanamex [NOT] relational operator {datanamew}
    {literal }
true condition imperative statement(s)
ELSE or OTHERWISE
false condition imperative statement(s)
ENDIF
```

Figure 77 AND Connector Format

```
IF DEPT EQ 201 AND
    JOBCLASS EQ 'C'
ACCEPT ELSE REJECT ENDIF
```

A record is accepted for output only if DEPT is equal to 201 and JOBCLASS is equal to C. If neither or only one of the conditions is true, the record is rejected.

```
IF FIELDA EQ FIELDC AND
    FIELDH (INW) GT FIELDJ AND
    FIELDM LT 721
    COUNTER=COUNTER+1
    GOTO PROCA
ELSE GOTO PROCB ENDIF
```

If all three of the compares are true, the COUNTER field is incremented by 1 and a branch to PROCA occurs. If none, one, or two of the compares is true, a branch to PROCB happens.

You can include series, range, or special tests (POSITIVE, NEGATIVE, NUMERIC) in your compound AND compares:

```
IF FIELDB EQ 511 THRU 513 AND
    FIELDC POSITIVE
NEXT ELSE REJECT ENDIF
```

Here, if FIELDB contains the value 511, 512, or 513 and FIELDC contains a positive value, execute the statement following ENDIF. Otherwise, if none or one of the compares is true, reject the record.

Combining Compound Compares

You can also combine AND and OR in the same conditional statement. For example, to select all Grade 7 pay-level employees in department 950 and all salaried (exempt) employees, code:

```
IF DEPT EQ 950 AND
    GRADE EQ '7'
    GOTO ABC ENDIF
IF EXEMPT EQ 'Y' NEXT
ELSE REJECT ENDIF
ABC:
```


Or, you can combine it in one conditional statement and obtain the same result:

```
IF DEPT EQ 950
  AND GRADE EQ '7' OR
  EXEMPT EQ 'Y' NEXT
ELSE REJECT ENDIF
```

When ANDs and ORs are combined in the same conditional statements, all AND combinations are evaluated first, left to right, before any OR combinations. In the previous example, a Grade 7 employee in department 950 results in a true condition. A non-Grade 7 employee in department 950 results in a false condition for the AND; but, if the employee was salaried, it would be a true condition for the OR and a true condition for the complete conditional statement.

To clarify the conditional statement for documentation purposes or alter the order of how simple conditions are combined, you can use parentheses to indicate how conditions are to be combined:

```
IF (DEPT EQ 950 AND GRADE EQ '7') OR
  EXEMPT EQ 'Y' NEXT
ELSE REJECT ENDIF
```

The result is the same as in the previous example; all salaried employees and all Grade 7 employees in department 950 are selected. If, however, you want to select only salaried or Grade 7 employees from department 950, code:

```
IF DEPT EQ 950 AND
  (GRADE EQ '7' OR EXEMPT EQ 'Y') NEXT
ELSE REJECT ENDIF
```

The OR condition is evaluated left to right before the AND condition. A non-exempt employee or anyone not in DEPT 950 causes a false condition.

If you are coding a complex IF statement and you are in doubt as to what order the condition is evaluated, use parentheses. All conditions within parentheses are evaluated together.

Conditions can be nested within parentheses:

```
IF DEPT EQ 950 AND
  (GRADE EQ '7' AND (EXEMPT EQ 'Y' OR HOURS LT 40)) NEXT
ELSE REJECT ENDIF
```

Salaried employees, Grade 7 employees in department 950, or employees working less than 40 hours per week in department 950 are selected.

When using parentheses in an IF statement, left and right parentheses must be matched or a validation error occurs.

Interval and Random Selection of Data

The IF SAMPLING command, followed by a unique application number, invokes interval or random selection of data in your VISION:Results program procedure logic. The format of this IF SAMPLING statement is as follows:

```
IF [NOT] SAMPLING nn
    true condition imperative statement(s)
ELSE or OTHERWISE
    false condition imperative statement(s)
ENDIF
```

Figure 78 IF SAMPLING Command

For a detailed description of IF SAMPLING, and how to define and use an interval or random selection request, see [Chapter 26: Using Random Selection](#) and [Chapter 27: Table and Array Handling](#).

Chapter 8: Structured Programming

The structured programming option of VISION:Results has been implemented for use by programmers and data processing installations that are involved in this methodology. All the generally accepted structures have been implemented in VISION:Results, and they use the traditional verbs associated with pseudo code.

The theory of structured programming supports three basic structures: sequence, selection, and iteration. Each structure must have only one entry point and only one exit. All three structures are fully supported by VISION:Results.

Sequence

The sequence structure is made up of one or more sequentially executed commands. Any of the VISION:Results commands, including PERFORM, can be combined to form a sequence structure.

GOTO, ACCEPT, and REJECT commands are disallowed when you are in structured mode.

Selection

The selection structure is used to control processing based on a specified condition. VISION:Results has two commands to accomplish the selection:

IF-THEN-ELSE-ENDIF
CASE-WHEN-ELSE-ENDCASE

The IF-THEN-ELSE-ENDIF combination is used to control processing based on a single predicate.

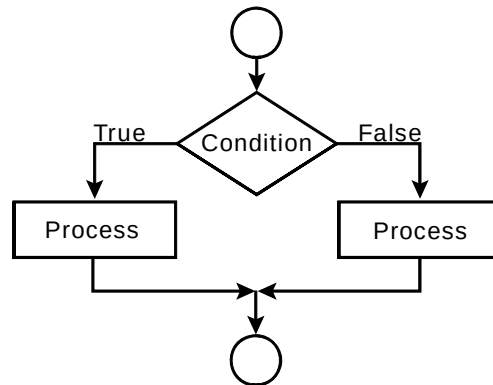


Figure 79 IF Structure

The CASE structure has been implemented to control processing based on multiple predicates.

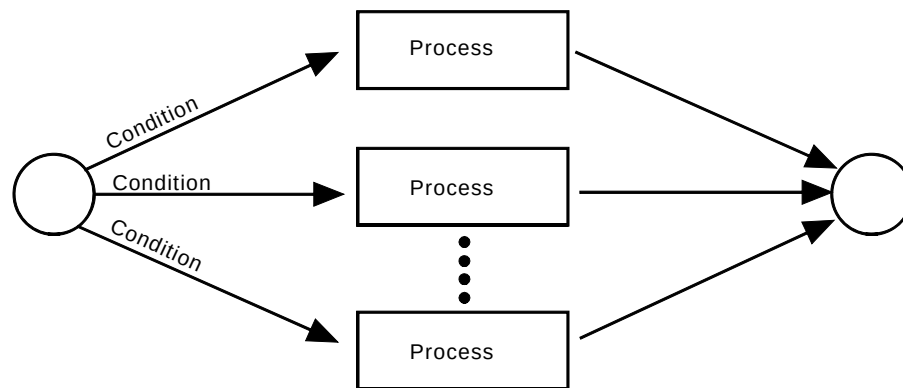


Figure 80 CASE Structure

In this structure, all predicates have the same subject but many different conditions. This structure is often used to improve readability and efficiency of the program. Suppose that processing is to be controlled based on a transaction code in the input record, and there are 50 of these codes. You could code 50 IF statements. The execution of this is inefficient and the listing is not easy to read. As an alternative, the CASE structure can be used.

Iteration

The iteration structure allows for looping, or the repeated execution of a segment of code. Two commands are available in VISION:Results to accomplish this:

DOWHILE
DUNTIL

The DOWHILE structure checks a specified condition and continues processing the loop while the condition is true. The condition is checked at the beginning of the loop, so it is possible to have zero iterations of the loop.

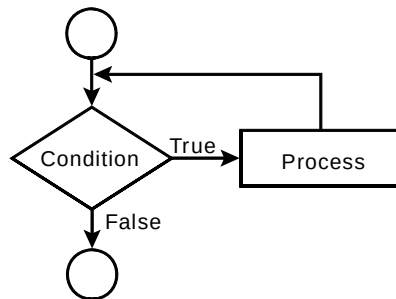


Figure 81 DOWHILE Structure

The DUNTIL structure checks a specified condition and continues processing the loop until the condition is true. The condition is checked at the end of the loop, so the loop is always executed at least once.

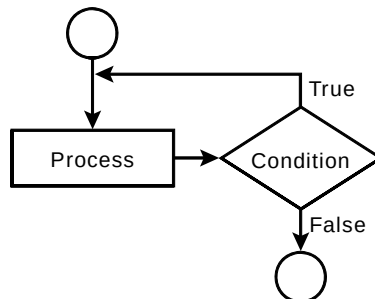


Figure 82 DUNTIL Structure

The various structures (sequence, selection, and iteration) can be combined and nested one within the other to accomplish any required task—the result is still a structured program. CA suggests that you indent the code for each structure that is subordinate to another to improve the readability of the resulting source listing.

Invoking Structured Mode

To use the structured features in VISION:Results, you must explicitly request it using the `OPTION` command or establish it as a default. The structured commands and nesting capability can lead to a complicated and difficult-to-understand program if used indiscriminately by an inexperienced programmer. To invoke this mode, specify the keyword `STRUCTURED` or `STRUCTURED2` on an `OPTION` statement at the beginning of your program. To establish either as a default, set the `DYLINSTL` macro parameter `PROGMOD` to `STRUCT` or `STRUCT2`, respectively.

```
[OPTION STRUCTURED | STRUCTURED2]
```

Figure 83 Invoke Structured Mode Option

This allows the use of the structured commands in the program as well as the nesting of `IF` statements. The option must be specified at the beginning of a program and remains in effect for the entire program. In a multiple-report run, the option remains in effect for all requests unless another `OPTION` statement is found in a subsequent request. You cannot mix modes in a single request, but you can change modes between requests in a multiple-report run.

The possible options are:

```
OPTION STRUCTURED
OPTION STRUCTURED2
OPTION CONVENTIONAL
OPTION USERDEFAULT
```

For descriptions of these formats in addition to other `OPTION` statements, see [Chapter 4: Using the `OPTION` Statement](#).

STRUCTURED

Allows the use of structured commands and nested `IF` statements. Disallows `GOTO`, `ACCEPT`, and `REJECT` commands.

STRUCTURED2

Allows the use of structured commands and nested `IF` statements. Also allows the use of `GOTO`, `ACCEPT`, and `REJECT` commands.

Users of VISION:Results Interface to DB2 and VISION:Results Interface to SQL/DS (VSE or CMS) should use `STRUCTURED2` mode or have the `DYLINSTL` macro parameter `STRUCGO` set to `Y` (`STRUCGO=Y`).

CONVENTIONAL

Sets mode to conventional VISION:Results. This disallows the use of structured commands and is the default mode for VISION:Results.

USERDEFAULT

Disallows the use of structured commands and some conventional report facilities. When in USERDEFAULT mode, VISION:Results makes some assumptions about any reports produced, thus reducing the amount of coding required by you. For a complete description of USERDEFAULT mode programming, see [Chapter 9: USERDEFAULT Mode Programming](#).

Restrictions

Structured programming has often been referred to as GOTO-less programming. This is not a restriction to structured techniques, but occurs if the structures are used correctly. To aid you in developing better programs, when STRUCTURED is requested, the GOTO command is disallowed. However, EXITDO, EXITCASE, and ITERATE can be used to circumvent the need for a GOTO. The VISION:Results commands ACCEPT and REJECT are also disallowed because they interfere with any iterative structure used. OPTION STRUCTURED2 allows you to use the GOTO, ACCEPT, and REJECT commands in a structured VISION:Results program.

Suggested Coding Standards

CA recommends that, when the STRUCTURED option is being used, some strict coding standards be followed. This is not a VISION:Results requirement. VISION:Results is completely free-form and the placement of commands, keywords, data names, and literals on the coding line has no meaning. However, adherence to coding standards improves the readability of the source listing, facilitates debugging, and makes future maintenance of the program much easier.

The following are suggested guidelines:

- Code only one command per line. If the command cannot fit on one line, it is continued on a second line or more, indented to show that it is part of the previous line.

```
LIST ACCTNO NAME TRANS
      BALANCE INTEREST
```

- Insert blank lines in the source to separate pieces of code.
- Code Reference points (tags) on a line by themselves.

```
LISTDET:
      LIST ACCTNO NAME TRANS
```

- Segment the program and use the NEWPAGE command prior to each segment so that each separate piece of code starts at the top of a new page. Each segment should be a discrete piece of code having only one entry and one exit. As a rule of thumb, no segment should contain more code than fits on one page (that is, 50 lines).

Choose meaningful names for data names and tag names. Also, in defining data names, prefix each data name with a constant to indicate to which file the data belongs. For example, all fields on an accounts receivable file might have their data names prefixed with the letters AR. For more information about the COPY or COPYE command, see [Chapter 15: Using the COPY Commands](#).

- Indent code in the program based on the structure to which the code belongs. Proper use of indentation graphically illustrates the relationship between the statements and structures. For example:
 - All tag names start in column 1.
 - All VISION:Results commands should start in column 4 or later. The exact location depends on whether the keyword is subordinate to a structure and, if so, which level.
 - All commands between an IF and ELSE should be indented four columns to the right of the IF. The ELSE should be lined up with the IF. All commands between ELSE and ENDIF should also be indented four columns. The ENDIF should be lined up with the IF and the ELSE.

```
IF condition
  command
  command
ELSE
  command
ENDIF
```

This gives a very clear picture of the scope of the IF. This same rule also applies if an IF is nested within another IF.

```
IF condition
  command
  command
  IF condition
    command
    command
  ELSE
    command
  ENDIF
ELSE
  command
ENDIF
```

- All commands between a DOWHILE or DOUNTIL and ENDDO should be indented four columns to the right of the DO. The ENDDO should line up with DO. If another structure is coded subordinate to the DOWHILE or DOUNTIL, the same rules for indentation should be followed.

```
DOWHILE condition
  command
  DOUNTIL condition
    command
    IF condition
      command
    ELSE
      command
    ENDIF
  ENDDO
ENDDO
```


- When the CASE structure is used, ENDCASE should line up with CASE. WHEN and ELSE should be lined up one under the other and four columns to the right of CASE. The code between WHEN and the next WHEN, WHEN and ELSE, WHEN and ENDCASE, or ELSE and ENDCASE should be indented eight columns to the right of CASE.

```
CASE dataname
    WHEN condition
        command
        command
    WHEN condition
        command
    ELSE
        command
        command
ENDCASE
```

- The following keywords and associated objects should appear on lines by themselves, indented the appropriate amount: IF, ELSE, ENDIF, DUNTIL, DOWHILE, ENDDO, CASE, WHEN, ENDCASE.
- Most commands should fit on one line. If a command does not fit, the remaining code goes on a second or subsequent line and should be indented eight columns.
- VISION:Results entry point commands (ON END OF SORTING, ON END OF INPUT, ON CHANGE IN, and ON FINAL) should start in column 1, and commands following should start in column 4 or later.
- Although a well-written, structured program is much easier to follow than an unstructured one, adequate documentation is still essential to complete understanding of the program. Comments should be logically placed in the code, symmetrically aligned, and set off from the code by asterisks or other noncode symbols or blank lines.

Again, this list of coding standards is only suggested and in no way limits the free-form capabilities of the VISION:Results syntax.

Sequence Structure

All the executable VISION:Results commands can be used as part of a sequence structure, except GOTO, ACCEPT, and REJECT. These three commands are disabled when in STRUCTURED mode. They can be enabled in STRUCTURED mode by using the OPTION STRUCTURED2 statement.

The other command described here is PERFORM because it is used extensively in all structured programming.

PERFORM

The PERFORM command causes control to pass to another segment of VISION:Results code. When the segment has been executed, control returns to the statement following the PERFORM.

The format of PERFORM is:

```
PERFORM tagname1 [TO tagname2]
      [VARYING name FROM initial TO ending [BY increment]]
```

where tagname1 is the tag name that identifies the first instruction to be performed and tagname2 is the tag name that immediately follows the last instruction to be executed.

The PERFORM is TO a tag name. Any commands following the TO tag name are not executed. If TO is not coded, the statements starting at tagname1 are executed until a tag or ON END/CHANGE is encountered.

The VARYING keyword is used as a counter for the PERFORM statement. Starting from an initial value, you can PERFORM this statement until an end value. The name entry must be a field name. The initial, ending, and increment values can be either field names or literals and can contain up to nine digits to the right of the assumed decimal point. The increment can be positive or negative. If the BY increment value is not specified, the default is 1.

You can nest PERFORM statements. That is, when you are in a segment that is performed, you can also issue another PERFORM.

If you attempt, in detail logic, to PERFORM a segment of code following an ON CHANGE or ON FINAL, a validation error results.

If you are issuing a PERFORM in ON CHANGE or ON FINAL logic and the segment you are performing is prior to an ON CHANGE or ON FINAL (that is, it is considered detail time code), take care that you do not execute a LIST in the performed code. Either a validation error occurs, or you get a 550 abend during execution.

Selection Structure

VISION:Results has two selection structures—CASE and IF. For a description of the basic IF command, see [Chapter 7: Using the IF Command](#). It is reviewed here because it is used in structured programming and because you can nest IF statements in STRUCTURED mode. The CASE structure is valid only while in STRUCTURED mode.

IF

The basic IF format is:

```
IF condition
[THEN] {imperative}
{ELSE imperative}
ENDIF
```

The specified condition is evaluated. When the condition is true, the code between the IF and the ELSE is executed and then control passes to the command following the ENDIF. If the condition is false, the code between the ELSE and the ENDIF is executed and then control passes to the command following the ENDIF. Either the true logic or the ELSE and false logic can be omitted, as needed. The word THEN is treated as a noise word by VISION:Results. You can use it if you think it improves the readability of your program.

Placement of ENDIF

Every IF statement must have a corresponding ENDIF. If each IF is not paired with an ENDIF, a validation error results. The ENDIF is important because it limits the scope of the IF. Care should be taken in determining where to place the ENDIF. Consider the following example:

```
IF BALANCE GT 300
    BALANCE = BALANCE * 1.10
ELSE
    BALANCE = BALANCE * 1.05
ENDIF
LIST BALANCE
```

When BALANCE is greater than 300, it is increased by 10%. All BALANCE entries less than or equal to 300 are increased by 5%. The ENDIF terminates the logic controlled by the condition and all BALANCES are listed.

Now consider:

```
IF BALANCE GT 300
    BALANCE = BALANCE * 1.10
ELSE
    BALANCE = BALANCE * 1.05
    LIST BALANCE
ENDIF
```

In this case, the BALANCE entries greater than 300 are increased by 10%. All other BALANCE entries are increased by 5%. Because the LIST command precedes the ENDIF, it is under control of the IF. Only those BALANCE entries that are less than or equal to 300 and have been increased by 5% are listed. The placement of the ENDIF is significant and becomes even more so when nested IF statements are used.

Nesting IF

When running under the STRUCTURED option of VISION:Results, nested IF statements are permitted. That is, another IF-THEN-ELSE-ENDIF can be issued subordinate to an existing IF. There is no limit to the number of IF statements that can be nested one within the other. However, take care not to nest IF statements too deeply. Beyond two or three levels, it becomes very difficult to follow the logic when nesting is used.

VISION:Results always pairs the ELSE and ENDIF with the preceding active IF. The evaluation of any IF and the execution of the appropriate code depends on the placement of the IF with respect to a higher-level IF (before or after the higher level ELSE) and the evaluation of the condition associated with the higher-level IF. The proper use of indentation makes it much easier to understand nested IF statements.

Remember, every IF must have a corresponding ENDIF, and the placement of it is significant, whether you are dealing with a simple IF or a nested IF.

Review the following example:

```
IF BALANCE GT 300
  IF ACCOUNT EQ 'EO'
    BALANCE = BALANCE * 1.20
  ELSE
    BALANCE = BALANCE * 1.10
  ENDIF
ELSE
  IF ACCOUNT EQ 'AO'
    BALANCE = BALANCE * 1.15
  ELSE
    BALANCE = BALANCE * 1.05
  ENDIF
ENDIF
LIST BALANCE
```

All BALANCE entries are being increased. The amount of the increase depends on the current BALANCE and the particular account. First, the BALANCE is checked. If the BALANCE is greater than 300, the account is checked. EO accounts get a 20% increase, while everyone else with a BALANCE greater than 300 gets a 10% increase. Control then passes to the LIST statement.

For BALANCE entries less than or equal to 300, the account is also checked. AO accounts receive a 15% increase, while all other BALANCE entries less than or equal to 300 get a 5% increase. After applying the appropriate increase, control passes to the LIST statement.

EO accounts with BALANCE entries less than or equal to 300 get only a 5% increase, and AO accounts with BALANCE entries greater than 300 get a 10% increase.

Nested IF statements are powerful tools that can result in very efficient object code being produced. They can also cause a lot of logic problems if not used carefully.

IF Restrictions

There are certain VISION:Results commands that cannot be issued subordinate to IF. Any attempt to do so results in a validation error. The restricted commands are:

```
SORT
ON END OF INPUT
ON END OF SORTING
ON CHANGE IN
```

```

ON FINAL
LIST          (at control break or ON FINAL time)
tagname definition (reference point)

```

CASE

If you have a variety of possibilities to check for on a particular field, you could nest IF statements to handle the problem and produce efficient code. However, depending on the number of possibilities, this could result in a program that is very difficult to read. The CASE structure produces efficient code and a program that is very easy to follow. Its basic function is to select code to be executed based on different values for a single subject. Unlike IF, which gives you two possibilities (the condition is true or it is false), the CASE structure allows for unlimited possibilities.

The format of the CASE structure is:

```

CASE dataname [index]
  WHEN condition1
    imperative
  WHEN condition2
    imperative
  :
  :
  ELSE
    imperative
ENDCASE

```

Figure 84 CASE Structure Format

The first statement of a CASE structure consists of the command CASE, followed by a data name (the content of this data name controls the processing flow within the CASE structure). The data name can be indexed; this index is not carried over to the next CASE structure.

Next comes a series of WHEN statements (one for each condition), followed by the condition to be tested. Notice that the data name is not repeated with each WHEN statement; VISION:Results automatically uses the data name from the CASE statement. ELSE is an optional keyword that executes its logic if all previous WHEN conditions prove false; therefore, it is logically placed after the last WHEN statement. (Remember, when an ELSE statement is used within the CASE structure, either a WHEN logic or the ELSE logic always executes.)

The ENDCASE statement is required as the last statement of the structure. The CASE and ENDCASE statements enclose the entire CASE structure.

The following two examples show a simple CASE statement and a logically equivalent non-CASE statement. Both of these examples have the same meaning:

```

CASE TRANCODE
  WHEN EQ 'A'

IF TRANCODE EQ 'A'

```

You can code compound conditions, as well as list and range compares, as part of the WHEN; just remember to omit the first operand. VISION:Results gives you a validation error if you forget. For example:

```
CASE ACCOUNT
  WHEN EQ 'EO' 'IO' 'NA' THRU 'PE'
```

During execution, VISION:Results evaluates condition 1. If it is true, the imperative statements between condition 1 and the next WHEN are executed and then control passes to the first statement following the ENDCASE. If condition 1 is false, VISION:Results evaluates condition 2. If it is true, the appropriate statements are executed and then control passes to the statement following ENDCASE. If condition 2 is false, VISION:Results proceeds to evaluate the next condition, and so on. The ELSE indicates that the code following is to be executed only if all the preceding conditions are false.

ENDCASE terminates CASE processing. Every CASE statement must have a corresponding ENDCASE. The placement of the ENDCASE is significant in that it indicates the end of code that is being controlled by CASE, as well as the start of code that is outside the scope of the structure.

If used properly, CASE produces very efficient code. Once a true condition has been detected, the appropriate code is executed and then control leaves the structure. No further tests are done.

Consider this example:

```
CASE YR_OF_RELEASE
  WHEN EQ 96
    PERFORM REV TO REV_X
  WHEN EQ 95
    PERFORM NOREV TO NOREV_X
  ELSE
    PERFORM ALL_OTHER TO ALL_OTHR_X
ENDCASE
LIST TITLE REVIEW YR_OF_RELEASE
```

All movies released in 1996 are placed on a list for critical review and are eligible for annual awards. All those with release dates in 1995 are processed as not eligible for awards, and all others are processed in some other manner. Because the ENDCASE precedes the LIST statement, all movies, regardless of year of release, are listed. If the ENDCASE statement followed the LIST, only those movies released in years other than 1996 or 1995 would be listed.

Nesting and CASE

You can nest one CASE structure within another; however, careful use of indentation makes following the logic easier.

CASE Restrictions

There are certain VISION:Results commands that cannot be issued subordinate to CASE. Any attempt to do so results in a validation error. The restricted commands are:

```

SORT
ON END OF INPUT
ON END OF SORTING
ON CHANGE IN
ON FINAL
LIST      (at control break or ON FINAL time)
tagname definition  (reference point)

```

EXITCASE

EXITCASE is valid only if CASE is in effect. It causes control to pass to the first statement following the ENDCASE for the CASE structure that is currently active. Usually, if VISION:Results is processing subordinate to a WHEN, it passes control to the statement following ENDCASE when it encounters another WHEN. However, if a condition is detected that indicates processing should not continue in the current WHEN, EXITCASE can be useful. This is typically required if an error condition has been detected.

In the following example, year of movie release is controlling processing within the structure. If a movie released in the current year (1996) is not included in the list of movies to be reviewed, the record is printed along with an error indication and control exits from the CASE structure.

```

CASE YR_OF_RELEASE
  WHEN EQ 96
    IF REVIEW NE 'Y'
      MOVE 'REVIEW ERR' TO DYLPRTCOMM
      PRINT MOVIEREC
      EXITCASE
    ENDIF
    PERFORM REV TO REV_X
  WHEN EQ 95
    PERFORM NOREV TO NOREV_X
  ELSE
    PERFORM ALL_OTHER TO ALL_OTHR_X
ENDCASE
LIST TITLE REVIEW YR_OF_RELEASE

```

All records with a review error are listed along with all the other movies because EXITCASE passes control to the statement following the ENDCASE (in this example, the LIST statement).

Iterative Structure

VISION:Results provides two iterative structures, DOWHILE and DUNTIL. These two structures allow you to repeatedly execute a piece of code. For example, continue processing input records until there are no more or scan a data record until a particular value is found. Different circumstances dictate which of the two structures should be used.

DOWHILE

DOWHILE tells VISION:Results to continue processing in the loop while the specified condition is true. When the condition is false, VISION:Results exits from the loop. VISION:Results evaluates the condition at the beginning of the loop. The format of DOWHILE is:

```
DOWHILE condition
    imperative
ENDDO
```

where condition is any valid IF condition as described in [Chapter 7: Using the IF Command](#). VISION:Results checks the specified condition. If the condition is false, control passes to the statement following the ENDDO. When the condition is true, the commands between DOWHILE and ENDDO are executed and then control returns to the evaluation of the condition.

Consider this example:

```
INX=0
DOWHILE INX LT 60
    TOTAL=TOTAL+MONTH (INX)
    INX=INX+5
ENDDO
```

Assuming that MONTH is defined as a 5-byte field, this DOWHILE loop adds 12 contiguous 5-byte fields to produce TOTAL (sum of 12 month's data).

DOUNTIL

The DOUNTIL is similar to the DOWHILE. However, DOUNTIL continues processing in the loop as long as the specified condition is false. DOUNTIL evaluates the condition at the END of the loop; therefore, the commands specified following DOUNTIL are always executed at least once regardless of the value of the condition. The format of DOUNTIL is:

```
DOUNTIL condition
    imperative
ENDDO
```

where condition is any valid IF condition as described in [Chapter 7: Using the IF Command](#). VISION:Results executes the commands between the DOUNTIL and ENDDO. The specified condition is then evaluated. If the condition is true, control passes to the statement following the ENDDO. When the condition is false, VISION:Results executes the commands in the loop again and then evaluates the condition.

Consider this example:

```
INX=0
DOUNTIL INX NOT LT 60
    TOTAL=TOTAL+MONTH (INX)
    INX=INX+5
ENDDO
```


Because this code is set up to loop through 12 iterations, this loop produces exactly the same result as the DOWHILE loop coded in [DOWHILE on page 168](#). Note, however, that the condition specified on the DOUNTIL is the opposite of the condition specified on the DOWHILE.

Placement of ENDDO

The placement of ENDDO indicates the range of the loop. Any code between the DO and the ENDDO is executed on an iterative basis. Any code following the ENDDO is executed only after control passes from the structure. Every DOWHILE or DOUNTIL must have a corresponding ENDDO. If you forget to code the ENDDO, a validation error results.

Nesting and DOWHILE or DOUNTIL

You can nest a DO within a DO. The use of indentation makes it much easier to follow the logic.

For example:

```
READ ARFILE
DOWHILE EOFSTAT NE 'E'
  MOVE SPACES TO FIRSTNAME
  INX=0
  DOUNTIL INX GT 24
    IF NAME(INX) EQ SPACE
      EXITDO
    ELSE
      INX=INX+1
    ENDIF
  ENDDO
  MOVE NAME TO FIRSTNAME LENGTH INX
  LIST FIRSTNAME
  READ ARFILE
ENDDO
STOP
```

In this example, a file ARFILE is defined with a STATUS byte of EOFSTAT. A field NAME is defined on the file that references the first byte of a 25-byte name field. A WORKAREA, FIRSTNAME, is set up to hold the first name only. This first name is to be listed. A READ is issued for ARFILE to get the first record. The DOWHILE loop indicates that control should remain in the loop until all the records have been read from ARFILE, then STOP is executed. Within this loop, the NAME field is scanned for a blank and then everything up to the blank is extracted as the first name. This first name is then listed prior to reading the next record on ARFILE and continuing in the loop.

DOWHILE or DOUNTIL Restrictions

There are certain VISION:Results commands that cannot be issued while subordinate to DOWHILE or DOUNTIL. Any attempt to do so results in a validation error. The restricted commands are:

```
SORT
ON END OF INPUT
ON END OF SORTING
ON CHANGE IN
ON FINAL
LIST                (at control break or ON FINAL time)
tagname definition  (reference point)
```

As with any loop, you must ensure that the condition that is being tested is eventually met as a result of the execution of the loop.

EXITDO

EXITDO is valid only if a DO (DOUNTIL, DOWHILE) is in effect. It causes control to pass to the first statement following the ENDDO for the currently active DO. Normally, control passes to this statement when the loop has been completed based on the specified condition. In some cases, however, a condition is detected within the loop that indicates that processing should not continue within the loop. For this purpose, EXITDO can be used.

In the following example, columns 1 to 60 of the input record are being scanned to find the first blank. Once it has been found, there is no need to continue looping.

```
INX=1
DOUNTIL INX GT 60
    IF COLUMN(INX) EQ BLANK
        EXITDO
    ENDIF
    INX=INX+1
ENDDO
```

On exit from this loop, INX contains the column number of the first blank. If there are no blanks in columns 1 to 60, INX contains the value 61.

ITERATE

If, while in a DOWHILE or DOUNTIL loop, you want to skip lines of code from a certain point to the end of the loop without exiting from the loop, you can use the ITERATE command. Placed inside of the loop, when it is executed, control skips all following code and branches to the ENDDO statement of the loop.

```
INX=0
DOUNTIL INX GT 12
  INX=INX+1
  MOVE COLUMN(INX) TO PRTCOL
  IF PRTCOL EQ BLANK
    PRINT 'COLUMN CONTENTS ERROR'
    PRINT PRTCOL
    ITERATE
  ENDIF
  LIST PRTCOL
.
.
ENDDO
```

If PRTCOL is blank, print the contents and a message. Then, bypass the LIST statement (ITERATE), return to the beginning of the DOUNTIL statement, and start another loop.

Automatic Cycle

When using the STRUCTURED option, you need to be aware of the automatic cycle and its possible impact on structured code.

Cycle Without SORT

Note: In this flowchart, when VISION:Results reads all input files and writes all output files, this applies only to those files that do not have READ/WRITE commands.

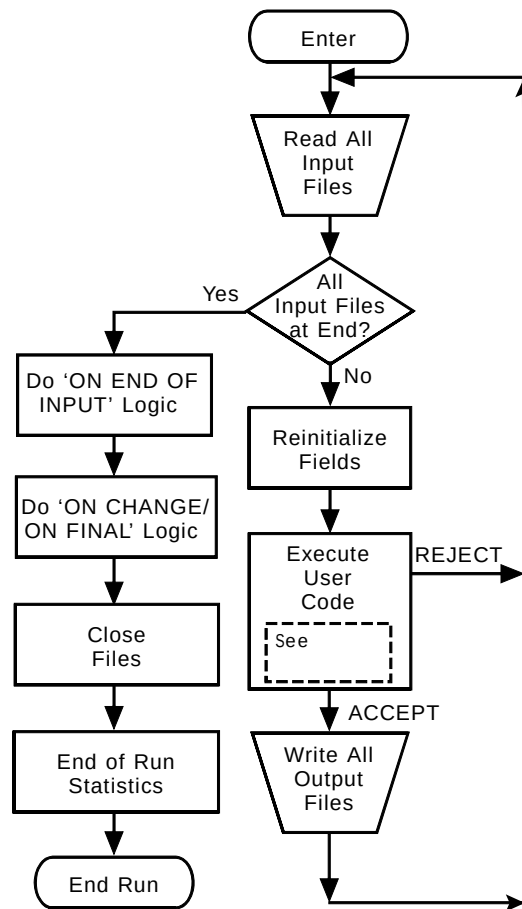


Figure 85 Automatic Cycle – No SORT

If there are any detail LIST statements, the following takes place whenever the first LIST statement is executed (before the detail report line is produced). If there are no detail LIST statements, the following takes place as part of the automatic cycle at the same point as output files are written.

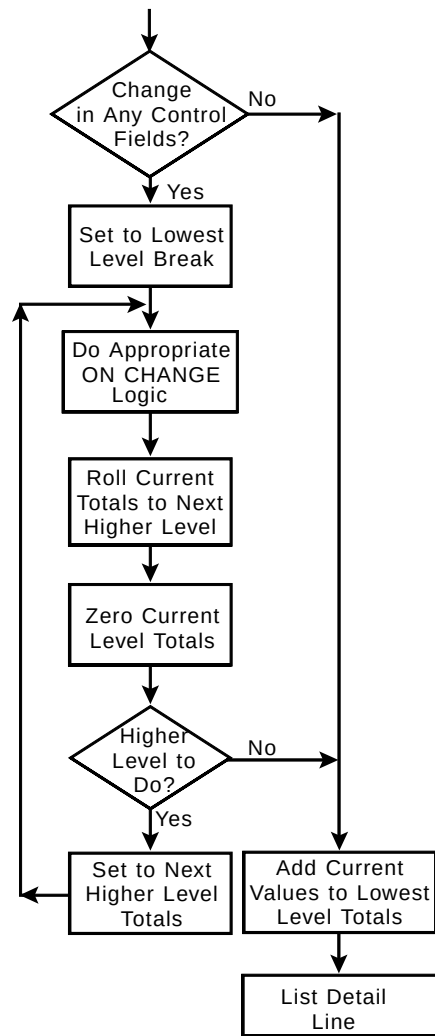


Figure 86 Automatic Cycle — LIST Statement Processing

The basic loop in VISION:Results consists of Read-Process-Write and then repeat. A record is read from each of the input files (that do not have a READ statement), your code is executed, and a record is written to each of the output files (that do not have a WRITE statement). This loop is repeated until all the input files are at end. When that occurs, VISION:Results does final processing and stops.

If you code a READ statement in the program, that particular file is read only when the READ statement is executed, not as part of the automatic cycle. Similarly, any output files that have WRITE statements specified for them are written only when the WRITE is executed, not as part of the automatic cycle.

In STRUCTURED mode, you cannot issue an ACCEPT or REJECT (unless you use OPTION STRUCTURED2), so the only way to enter the automatic cycle is to exit at the bottom of the program. If you have set the program up with a DO indicating a loop that should be executed until the input is exhausted, the automatic cycle is not entered until all the input has been read. When this technique is used, you must issue READs and WRITEs for all your files.

VISION:Results automatically terminates processing as soon as all input files are at end. This includes automatically read files as well as those processed by a READ command. Once all input files are at end, you do not get control again unless ON END OF INPUT has been specified. This should not cause a problem. However, if you have coded a DO loop specifying "process until files are at end," you are not able to execute wrap-up logic unless you use ON END OF INPUT or ON FINAL.

Remember, it is when all input files are at end that VISION:Results terminates. Files processed in a skip sequential or random manner never go to end. Thus, with any of these files in the program, VISION:Results does not terminate automatically. In this case, you get control when the loop is complete and you must issue your own STOP.

If you have only sequentially read files in your program but want to control when to terminate processing, you can put a DUMMY file in your program.

```
FILE INFILE DUMMY
```

Because a DUMMY file is never read, it can never go to end. If it does not go to end, there is always at least one file that is not at end and, therefore, VISION:Results does not terminate automatically. You get control after all input files are at end, but you must STOP the program yourself.

The logic to handle reinitializing fields is part of the automatic cycle. If the program is not going through the automatic cycle because of a DO, the REINIT function never executes.

Recommendations without SORT:

- Use READ and WRITE commands for all files when in STRUCTURED mode.
- Do not use REINIT. If you need to have fields re-initialized, insert code to do it at the beginning of your loop.
- If you have special processing requirements when all files are at end, make sure that you have a DUMMY file and a STOP statement in your program.

Cycle with SORT

Note: In this flowchart, when VISION:Results reads all input files and writes all output files, this applies only to those files that do not have READ/WRITE commands.

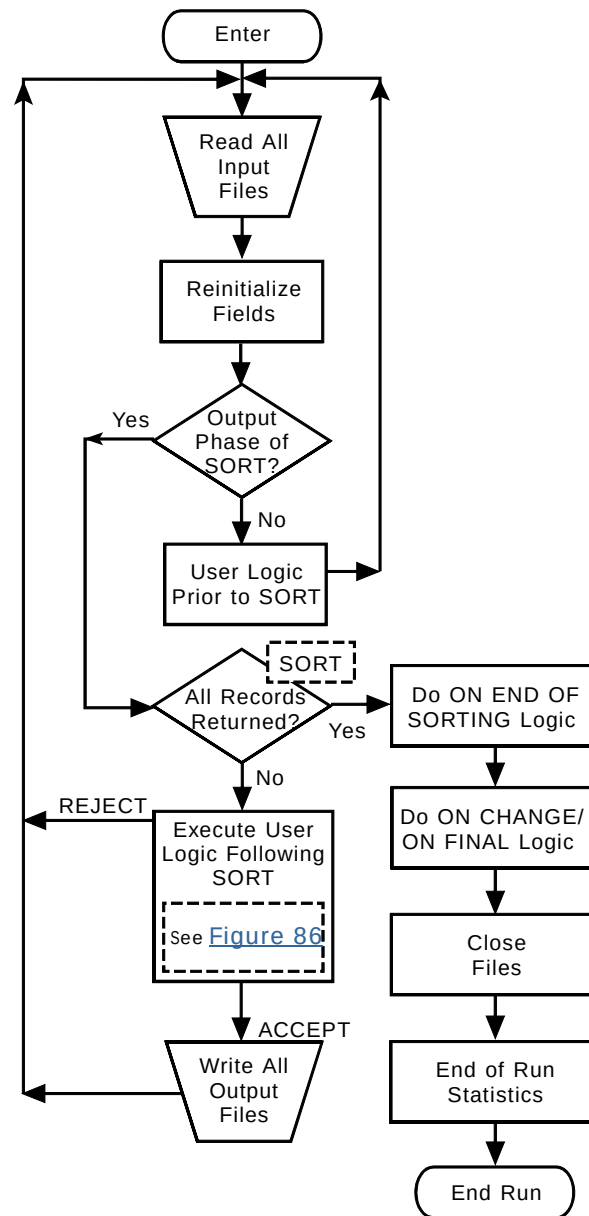


Figure 87 Automatic Cycle — SORT

The automatic cycle without SORT looks like a single circle: Read-Process-Write and back to the read. The automatic cycle with SORT can be considered as a double loop or a figure 8. VISION:Results loops in the top half of the figure 8 during the input phase of the SORT and in the bottom half during the output phase of the SORT. The UNTIL clause on the SORT statement or end of file on the file to be sorted determines when VISION:Results changes from one loop to the other.

Each record is passed to SORT during the input cycle and then VISION:Results goes through the automatic cycle and enters your logic at the top again.

Records are retrieved on the output phase of the SORT only as part of the automatic cycle, so you must not set up a loop after the SORT that prevents code from entering the automatic cycle.

VISION:Results does not allow the SORT command to be part of a DO loop. You probably want to set up a loop ahead of the SORT statement to select records for sorting and possible reformatting.

When SORT is used, VISION:Results terminates the program when all records have been processed (when the sort program indicates that it is finished). All input files being at end is not considered when terminating a VISION:Results program with SORT. Therefore, you never need a DUMMY file if you have a SORT in your program.

If you need special wrap-up logic when all the records have been returned from SORT, you can use the ON END OF SORTING entry point to get control to do your processing. You have to issue your own STOP if you do this.

Processing always goes through the automatic cycle during both input and output phases of the SORT. REINIT can be used if needed in this case.

Recommendations with SORT:

- Use READ and WRITE commands for all files when in STRUCTURED mode.
- If you have special processing requirements when SORT has completed, make sure that you have an ON END OF SORTING entry and a STOP statement in your program.

Structured Programming Techniques

The following are some approaches to circumvent the use of commands that cannot be issued while using structured programming.

Select Records for SORT

Because you cannot REJECT records that you do not want to sort, you must READ past records that do not meet the selection criteria. You can use either a DOWHILE or a DOUNTIL to accomplish this. If you use the DOWHILE, you must code a READ followed by a DOWHILE specifying the inverse of the selection criteria. Following the DOWHILE, you have another READ of the file.

```
READ INPFILE
DOWHILE NOT selection criteria
    READ INPFILE
ENDDO
SORT INPFILE ....
```

A record is read. Because DOWHILE does the test at the beginning of the loop, that record is checked against the selection criteria. (Note the use of NOT.) If the record meets the criteria (that is, NOT selection criteria is false), control passes to the SORT statement. VISION:Results then exits through the automatic cycle, re-enters the program at the top of the program, and reads the next record on the file. For any record not meeting the selection criteria (that is, NOT selection criteria is true), control remains in the loop and another record is read. This continues until end of file is reached on the file and VISION:Results switches to the output phase of the sort.

If the DOUNTIL is used, you code DOUNTIL specifying the selection criteria. This is followed by a READ statement, an ENDDO, and then the SORT.

```
DOUNTIL selection criteria
    READ INPFILE
ENDDO
SORT INPFILE ...
```

With DOUNTIL, the test is done at the end of the loop. When the program is entered, the file is read and then the record is tested to determine if it meets the selection criteria. If it does, control passes to the SORT statement. If it does not meet the criteria, control remains within the loop and the next record on the file is read. This loop continues until a record is read that meets the criteria. Once a record has been passed to SORT, VISION:Results exits through the automatic cycle and re-enters the program at the top. In this case, it enters the DOUNTIL loop again and reads the next record.

With either the DOWHILE or DOUNTIL solution, you do not have to worry about an infinite loop. Because there is a SORT in the program, when the file goes to end of file, control automatically transfers to the output phase of the sort.

ACCEPT

The ACCEPT command is not valid in STRUCTURED mode. Before inserting code in the VISION:Results program to duplicate this function, be aware of what ACCEPT is doing. When ACCEPT is issued, all output files that do not have WRITE specified are written. If there is no detail LIST statement, totaling and control break checking is done. Control then enters at the beginning of your code after all input files that do not have READ specified have been read. If there is a SORT in the program, ACCEPT following the SORT statement causes another record to return from SORT.

If you want to issue ACCEPT to cause records to be written, to cause records to be read, or to cause control to return to the top of the program, code a large loop in your program that has READ statements at the top and WRITE statements at the bottom.

```
DOWHILE not all files at end
  READ file1
  READ file2
  READ file3
  .
  .
  .
  WRITE file4
  WRITE file5
  WRITE file6
ENDDO
```

This simulates ACCEPT, but does not go through the automatic cycle.

If you need to go through the automatic cycle for totaling or control break checking when there is no detail LIST statement, or to retrieve another record from SORT, you must code the program to allow control to fall into the automatic cycle. VISION:Results exits through the automatic cycle whenever control passes beyond the last coded statement or whenever control reaches an ON END OF INPUT, ON END OF SORTING, ON CHANGE, or ON FINAL. Remembering this, you can code a DOWHILE or DOUNTIL loop that spans the program up to one of these entry points and then, when you want to exit to the automatic cycle, code EXITDO.

```
DOUNTIL ENDFLAG EQ 'X'
  READ FILE1
  .
  .
  .
  IF condition1
    EXITDO
  ENDF
  .
  .
ENDDO
ON END OF INPUT
STOP
```

In this example, the DOUNTIL condition is one that is never true; this is an infinite loop unless some other condition occurs causing control to pass from the loop. Control remains in this loop until all records on FILE1 are read. Control then passes to the ON END OF INPUT entry point and STOP is executed. Whenever EXITDO is executed, control passes from the loop. The first statement following the loop is the ON END OF INPUT; therefore, control exits through the automatic cycle and re-enters at the top of the program. For an additional example of this technique where SORT is involved, see [Match Files Where One File Must Be Sorted on page 184](#).

REJECT

The REJECT command is not valid in STRUCTURED mode. REJECT exits through the automatic cycle but bypasses the automatic writing of output files and any control break checking and totaling that might otherwise be done. Then, it reads input files (that do not have READ commands) prior to entering at the top of the program. This can easily be simulated in STRUCTURED mode by bypassing WRITE commands and continuing in a READ loop to bypass unwanted records. In CONVENTIONAL mode, the DOUNTIL loop might be replaced by:

```
IF ACCOUNT NE 'NA' 'EO' THRU 'IO' REJECT ENDIF
```

GOTO

The GOTO command is not valid in STRUCTURED mode. In most cases, GOTO is not needed in a structured program. Careful use of nested IF statements, DO loops, and PERFORM statements should eliminate the need for GOTO. While subordinate to DOWHILE/DOUNTIL, EXITDO can be used to GOTO the first statement following ENDDO. Within a CASE structure, EXITCASE can be considered as a GOTO the code following ENDCASE.

Structured Programming Examples

The following examples illustrate the use of structured programming.

Example 1 Sequential File Update

```
OPTION STRUCTURED

FILE OLDMAST FB 77 1540 STATUS MAST_EOF
OM RECORD 77
OM_SOCSEC 9 1

FILE TRANFIL FB 80 800 STATUS TRAN_EOF
TR RECORD 80
TR_SOCSEC 9 1
TR_LASTNAME 15 20
TR_ADDRESS 35
TR_SALARY 7 NU 2
TRANCODE 1 80

FILE NEWMAST OUTPUT FROM NEWMAST FB 77 1540
NM RECORD 77
NM_LASTNAME 15 20
NM_ADDRESS 35
NM_SALARY 7 NU 2

WORKAREA
OM_KEY 9 ;MASTER KEY HOLD
TR_KEY 9 ;TRANSACTION KEY HOLD
NEW_SW 1 VALUE ' '
DEL_SW 1 VALUE ' '

NEWPAGE
MAINLINE:
PERFORM READ MAST TO RDM_EX
PERFORM READ TRAN TO RDT_EX
DOWHILE MAST_EOF NOT EQ 'E' OR TRAN_EOF NE 'E'
  IF OM_KEY LT TR_KEY
    PERFORM WRITE MAST TO WTM_EX
    IF NEW_SW EQ 'N'
```

Figure 88 Sequential File Update (Page 1 of 2)

```
        MOVE SPACE TO NEW_SW
      ELSE
        PERFORM READ_MAST TO RDM_EX
      ENDIF
    ELSE
      IF OM_KEY EQ TR_KEY
        CASE TRANCODE
          WHEN EQ 'A'
            MOVE 'BAD ADD' TO DYLPRTCOMM
            PRINT TR_RECORD
          WHEN EQ 'C'
            PERFORM CORRECTION TO COR_EX
          WHEN EQ 'D'
            MOVE 'D' TO DEL_SW
            MOVE 'DELETED' TO DYLPRTCOMM
            PRINT TR_SOCSEC
          WHEN EQ 'U'
            PERFORM UPDATE TO UPD_EX
          ELSE
            MOVE 'BAD TRAN' TO DYLPRTCOMM
            PRINT TR_RECORD
        ENDCASE
        PERFORM READ_TRAN TO RDT_EX
      ELSE
        IF TRANCODE EQ 'A'
          PERFORM ADDITION TO ADD_EX
        ELSE
          MOVE 'NO MATCH' TO DYLPRTCOMM
          PRINT TR_RECORD
        ENDIF
        PERFORM READ_TRAN TO RDT_EX
      ENDIF
    ENDDO
  STOP
ON END OF INPUT
IF NEW_SW EQ 'N'
  WRITE NEWMAST
ENDIF
STOP
NEWPAGE
READ_MAST:
  READ OLDMAST
  MOVE OM_SOCSEC TO OM_KEY
  IF MAST_EOF NE 'E'
    MOVE OM_RECORD TO NM_RECORD
  ENDIF
RDM_EX:

READ_TRAN:
  READ TRANFIL
  MOVE TR_SOCSEC TO TR_KEY
RDT_EX:

WRITE_MAST:
  IF DEL_SW EQ 'D'
    MOVE SPACE TO DEL_SW
  ELSE
    WRITE NEWMAST
  ENDIF
  IF NEW_SW EQ 'N'
    MOVE OM_RECORD TO NM_RECORD
    MOVE OM_SOCSEC TO OM_KEY
  ENDIF
WTM_EX:
ADDITION:
  MOVE 'N' TO NEW_SW
  MOVE TR_RECORD TO NM_RECORD
  MOVE TR_KEY TO OM_KEY
ADD_EX:
CORRECTION:
  IF TR_LASTNAME NE SPACES
    MOVE TR_LASTNAME TO NM_LASTNAME
  ENDIF
  IF TR_ADDRESS NE SPACES
    MOVE TR_ADDRESS TO NM_ADDRESS
  ENDIF
  IF TR_SALARY NE 0
    MOVE TR_SALARY TO NM_SALARY
  ENDIF
COR_EX:

UPDATE:
  NM_SALARY = NM_SALARY + TR_SALARY
UPD_EX:
```

Figure 88 Sequential File Update (Page 2 of 2)

Example 2 Random File Update

```

OPTION STRUCTURED

FILE KMASTER KSDS IO RANDOM SEARCHKEY F 77
  KEYLEN 9 ERASE ERASEFLG STATUS KSDS_STAT
  KM RECORD 77
  KM SOCSEC 9 1
  KM_LASTNAME 15 20
  KM_ADDRESS 35
  KM_SALARY 7 NU 2

FILE TRANFIL FB 80 800 STATUS TRAN_EOF
  TR RECORD 80
  TR SOCSEC 9 1
  TR_LASTNAME 15 20
  TR_ADDRESS 35
  TR_SALARY 7 NU 2
  TRANCODE 1 80

MAINLINE:
  PERFORM READ_TRAN TO RDT_EX
  DOWHILE TRAN_EOF NOT EQ 'E'
    PERFORM READ_KSDS TO RDK_EX
    IF KSDS_STAT EQ 'Y'
      IF TRANCODE EQ 'C'
        PERFORM CHANGE_RTN TO CHG_EX
      ELSE
        IF TRANCODE EQ 'D'
          MOVE 'E' TO ERASEFLG
          PERFORM WRITE_KSDS TO WTK_EX
        ELSE
          PERFORM BAD_TRAN TO BAD_EX
        ENDIF
      ENDIF
    ELSE
      IF TRANCODE EQ 'A'
        MOVE TR_RECORD TO KM_RECORD
        PERFORM WRITE_KSDS TO WTK_EX
      ELSE
        PERFORM BAD_TRAN TO BAD_EX
      ENDIF
    ENDIF
    PERFORM READ_TRAN TO RDT_EX
  ENDDO
  STOP
NEWPAGE
READ_KSDS:
  MOVE TR_SOCSEC TO SEARCHKEY
  READ KMASTER
RDK_EX:

READ_TRAN:
  READ TRANFIL
RDT_EX:

WRITE_KSDS:
  WRITE KMASTER
WTK_EX:
CHANGE_RTN:
  IF TR_LASTNAME NE SPACES
    MOVE TR_LASTNAME TO KM_LASTNAME
  ENDIF
  IF TR_ADDRESS NE SPACES
    MOVE TR_ADDRESS TO KM_ADDRESS
  ENDIF
  IF TR_SALARY NE 0
    MOVE TR_SALARY TO KM_SALARY
  ENDIF
CHG_EX:

BAD_TRAN:
  MOVE 'BAD_TRAN' TO DYLPRTCOMM
  PRINT TR_RECORD
BAD_EX:

```

Figure 89 Random File Update

Example 3 Two-File Merge

```
OPTION STRUCTURED

FILE FILE1 STATUS F1_EOF FB 210 1680
  F1_RECORD 210
  F1_KEY    7   3

FILE FILE2 STATUS F2_EOF FB 210 1680
  F2_RECORD 210
  F2_KEY    7   3

FILE MERGED OUTPUT FROM MERGED FB 210 1680
  OUTREC 210
PROC:
  PERFORM READ_F1 TO RD1_EX
  PERFORM READ_F2 TO RD2_EX

  DOWHILE F1_EOF NOT EQ 'E' OR F2_EOF NOT EQ 'E'
    IF F1_KEY LT F2_KEY
      MOVE F1_RECORD TO OUTREC
      PERFORM WRITE_OUT TO WRT_EX
      PERFORM READ_F1 TO RD1_EX
    ELSE
      IF F1_KEY GT F2_KEY
        MOVE F2_RECORD TO OUTREC
        PERFORM WRITE_OUT TO WRT_EX
        PERFORM READ_F2 TO RD2_EX
      ELSE
        MOVE F1_RECORD TO OUTREC
        PERFORM WRITE_OUT TO WRT_EX
        MOVE F2_RECORD TO OUTREC
        PERFORM WRITE_OUT TO WRT_EX
        PERFORM READ_F1 TO RD1_EX
        PERFORM READ_F2 TO RD2_EX
      ENDIF
    ENDIF
  ENDDO
  STOP
NEWPAGE
READ_F1:
  READ FILE1
  RD1_EX:

READ_F2:
  READ FILE2
  RD2_EX:

WRITE_OUT:
  WRITE MERGED
  WRT_EX:
```

Figure 90 Two-File Merge

Example 4 Scan of PDS Members (z/OS)

```

OPTION STRUCTURED
FILE FILE1 FB 80 PDS INAREA1 MEMBER INMEM1 STATUS INSTAT
INREC1 80
SCANREC 10 7

WORKAREA
PRTMEM 8

READDIR FILE1

DOWHILE INSTAT NE 'E'
  READMEM FILE1
  DYLCOUNT1 = DYLCOUNT1 + 1
  PRINT INSTAT
  DOWHILE INSTAT NE 'E'
    INX = 0
    DOUNTIL INX GE 54
      IF SCANREC(INX) EQ 'UNIT=SYSDA'
        LIST SUPPRESS INMEM1 INREC1
        EXITDO
      ELSE
        INX = INX + 1
      ENDIF
    ENDDO
  READMEM FILE1
  ENDDO
  INSTAT = ' '
  READDIR FILE1
  ENDDO
  STOP
T1 'COMPUTER ASSOCIATES' WITH 2 AFTER
T1+80 DYLDATE
FIN

```

Figure 91 Scan of PDS Members (z/OS)**Example 5 SELECT for SORT and REPORT**

```

OPTION STRUCTURED
FILE ARFILE FB 352 5280
ACCOUNT 2 182 NAME 25 85 TRANS 7 4 BALANCE 5 170 PD 2 A
WORKAREA COMMENT 3 (BALANCE'OVER $300.00) VALUE ' ' REINIT

CONTROL ACCOUNT
  DOUNTIL ACCOUNT EQ 'NA' 'EO' THRU 'IO'
    READ ARFILE
  ENDDO
  SORT ARFILE USING ACCOUNT TRANS
  IF BALANCE GE 300
    MOVE '***' TO COMMENT
  ENDIF
  LIST SUPPRESS ACCOUNT TRANS NAME BALANCE COMMENT

ON CHANGE IN ACCOUNT
  LIST SUM BALANCE WITH 2 BEFORE AND 2 AFTER

ON FINAL
  LIST SUM BALANCE WITH 2 AFTER
  LIST 'NUMBER OF RECORDS PRINTED:' AT ACCOUNT TALLY AT BALANCE

T1 'VISION:Results REPORT 8' WITH 2 AFTER
T1+30 DYLDATE
T1+90 DYLPAGE
FIN

```

Figure 92 SELECT for SORT and REPORT

Example 6 Match Files Where One File Must Be Sorted

```
OPTION STRUCTURED

FILE FILEA  FB 352 5280 STATUS FILEASTAT
READKEY    7  4
.
.
FILE FILEB  FB 352 10560
SORTKEY    7  8
.
.
WORKAREA
SORTSTAT  1
.
.
  READ FILEB
  SORT FILEB

ON ONE
  READ FILEA
ENDONE
  DOWHILE FILEASTAT NE 'E' OR
    SORTSTAT NE 'E'
    CASE SORTKEY
      WHEN EQ READKEY
        PERFORM MATCHIT TO MATCHITX
        READ FILEA
        EXITDO
      WHEN LT READKEY
        PERFORM NEWREC TO NEWRECX
        EXITDO
      ELSE
        PERFORM COPMAST TO COPMASTX
        READ FILEA
    ENDCASE
  ENDDO
NEWPAGE
  MOVE 'E' TO SORTSTAT
  DOWHILE FILEASTAT NE 'E'
    PERFORM COPMAST TO COPMASTX
    READ FILEA
  ENDDO
  STOP

MATCHIT:
; LOGIC FOR HANDLING TWO MATCHED RECORDS
MATCHITX:

NEWREC:
; LOGIC FOR HANDLING A SORTED RECORD
; WITH NO MATCHING FILE RECORD
NEWRECX:

COPMAST:
; LOGIC FOR HANDLING A FILE RECORD
; WITH NO MATCHING SORTED RECORD
COPMASTX:
FIN
```

Figure 93 Match Files Where One File Must Be Sorted

Chapter 9: USERDEFAULT Mode Programming

USERDEFAULT Option

The USERDEFAULT mode assumes certain defaults to simplify use of VISION:Results. For novice programmers, the USERDEFAULT is a quick and easy method to produce lists and reports.

The USERDEFAULT option requires a minimum of four statements to produce a list or report. The four statements are:

- OPTION statement
- FILE statement
- Field definition statement
- LIST statement

Invoking USERDEFAULT Mode

Note: The keyword USERDEFAULT can be used only with the OPTION command.

Specify the USERDEFAULT mode using the OPTION statement. When used, the OPTION statement must precede all other statements. For exceptions to this rule, see [Chapter 4: Using the OPTION Statement](#).

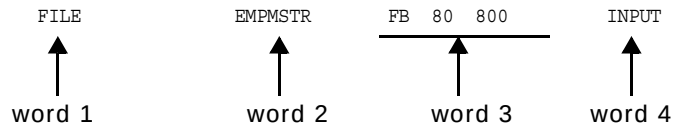
The OPTION statement specifying the USERDEFAULT mode contains two words—OPTION and USERDEFAULT.

```
OPTION USERDEFAULT
```

FILE

The FILE statement immediately follows the OPTION statement. It provides necessary information about file characteristics and processing mode, including record format, length, block size, input or output mode, and sequential or random access

The FILE statement defines the attributes of the input file you want to read and report on.



This statement contains four words:

- The first word, FILE, is the command.
- The second word, EMPMSTR, is the name of the file you want to process (in z/OS, it is called the ddname). The file name is generally arbitrary (you can choose it); however, it is directly associated with the job control language statements (JCL) in your program. JCL for the FILE statement is discussed in [Appendix B: Job Control Language](#). The name of your file is positional; that is, it must follow the command, FILE.
- The third word, FB 80 800, is sometimes referred to as a keyword phrase. This describes the format, the record size, and block size of the file, respectively. In this instance, the record is a fixed block, 80-byte record and there are 10 records to each block.
- The fourth word in the FILE statement is INPUT, which is the file you want to read and report on. This file already exists. INPUT is a default keyword for a FILE statement; therefore, it is not required. The FILE statement could have been stated as follows:

```
FILE EMPMSTR FB 80 800
```

Record Format

FB is the keyword for FIXED BLOCKED, which means that the file contains records that are equal in length and grouped into even multiples called blocks. These multiples are divisible by the record size. For example, a record size of 100 bytes and block size of 1000 bytes means there are 10 records to a block.

Record Size

Immediately following the record format is the record size, 80, which must follow the keyword FB. The record size is the length in bytes of each record on the EMPMSTR file.

Block Size

Block size is subordinate to record size. That is, the block size cannot be specified unless it immediately follows the record size. In the example, the block size is 800. This means that ten 80-byte records (80x10) make up one block.

Field Definitions

The third statement is the field definition statement. You can use more than one. Along with the name of the field, you need to give its size. In this example, you are defining only those fields you want to list or report on.

```
DIV      1
DEPT     3
EMPNO    5
NAME     20
```

In this example, you are defining four fields in field definition statements: DIV, DEPT, EMPNO, and NAME. These are the names you have given the fields. Data names can be from 2 to 50 characters long. The default type is character. For a complete list of field types, see [Edit Codes on page 313](#).

Size/Length

The number immediately following each data name is the size or length of the field in the record, which is mandatory. DIV is 1 byte long, DEPT is 3 bytes long, EMPNO is 5 bytes long, and NAME is 20 bytes long.

Unless otherwise specified, VISION:Results assumes that the beginning location of the first field definition is position 1 of the record.

Length/Size	1	3	5	20
Data name	DIV	DEPT	EMPNO	NAME
Location in the record	1	2	5	10

Because DIV is 1 byte long (assuming position 1 in the record), the beginning location for DEPT would be position 2.

Procedure Logic

After you define the field definitions for the file, you can begin to write the procedure logic. The procedure logic tells VISION:Results what you want it to do (such as read the file, sort the records, and report on the fields as specified).

VISION:Results automatically reads the EMPMSTR file. You can do it yourself, but this option should be used only in the other two modes of operation. Because there are no selection (IF) statements in the procedure logic, all of the EMPMSTR records are read and reported on.

You generate reports with the LIST command.

LIST

The last statement in the example defines the actual report. This is the only statement specified in the procedure logic.

```
LIST DIV DEPT EMPNO NAME
```

The LIST command tells VISION:Results *what* should be printed and *where* it prints.

The LIST statement in the example has five words. The first word is the command, LIST, which defines the contents of the report line to be printed. The remaining four words are the data names of fields that are to appear on the report. The first field that prints (left to right) is DIV, followed by DEPT, EMPNO, and NAME.

```
01/15/01PAGE 1
```

DIV	DEPT	EMPNO	NAME
1	103	10692	W A ALTMAN
1	103	11967	R SPRUCE
1	103	12352	B B CRUTHERS
1	103	13126	J R LAWRY
1	209	00178	J WELL
2	108	01273	W P MATHERS
2	108	02289	S ALTENBERG
2	108	02413	M R ROSEN
2	108	03523	L A MARTIN
2	126	03007	C R SMITH
2	126	03106	B BLAKE
2	127	03132	F A CALLEY
3	106	04142	P R JONES
3	106	23415	S L HALPER
3	110	87214	S KRAN2
3	110	30241	P LING
3	110	00121	J NEWMAN

17 RECORDS LISTED

USERDEFAULT Considerations

With the exception of the items below, all commands and keywords are available to users running in USERDEFAULT mode.

- SUM and NOSUM keywords can be used only with field definitions.
- SUM and NOSUM keywords are ignored by other modes of operation.
- SUM and SUMn cannot be used with any commands in procedure logic when running in USERDEFAULT mode.
- ON CHANGE IN and ON FINAL cannot be used in USERDEFAULT mode.
- Summary reporting is not allowed when running in USERDEFAULT mode.
- A maximum of eight Tn (title) statements is available in USERDEFAULT mode.

Examples of Programming in USERDEFAULT Mode

The programs on the following pages use two unique input files — an employee master file (EMPMSTR) and an accounts receivable file (ARFILE). Examples of programming in the USERDEFAULT mode are introduced, each one building on the previous one. Commands and keywords are introduced and defined as you progress.

The EMPMSTR File

Length or Size	1	3	5	20	4	3	20	6	18
Data name	DIV	DEPT	EMPNO	NAME	FILLER	HOURS	SUPERVISOR	HIREDATE	FILLER
Location in the Record	1	2	5	10	30	34	37	57	63

Data name	Size (length)	Type	Description
DIV	1		Division number
DEPT	3		Department number
EMPNO	5		Employee number
NAME	20		Employee name
FILLER	4		Not used
HOURS	3	Numeric	Hours worked; maximum 99.9
SUPERVISOR	20		Employee's immediate supervisor
HIREDATE	6	Numeric	Date employee was hired
FILLER	18		Not used

Example 1 Generate a Report Using the EMPMSTR

Produce a listing of all employees on the employee master file (EMPMSTR). Print the division number, the department number, the employee number, the employee name, and hours worked. Define only those fields needed to produce the report. Use the data names defined in the table starting on page [189](#).

```
OPTION USERDEFAULT
FILE EMPMSTR FB 80 800
    DIV      1
    DEPT     3
    EMPNO    5
    NAME     20
```

```
        FILLER  4
        HOURS   3  NU   1  E
LIST DIV DEPT EMPNO NAME HOURS
```

Each VISION:Results statement used to produce this report is reviewed.

OPTION Statement

The OPTION statement requires use of USERDEFAULT. It must be the first statement in the program. A minimum of one space is required between the command and the keyword.

```
OPTION USERDEFAULT
```

FILE Statement

At least one FILE statement is always required. Its purpose is to define the file name and its attributes.

```
FILE EMPMSTR FB 80 800
```

VISION:Results syntax has certain defaults, which lessen the coding task. You need not code a keyword that is also defined as the default for a command.

The employee master file, EMPMSTR, is an input file. The default on a FILE statement is input; therefore, you can exclude the keyword INPUT. If provided, the FILE statement above would look like:

```
FILE EMPMSTR FB 80 800 INPUT
```

Field Definition Statements

These statements define the fields in the record of the input file that appears on the report. The field definitions for any file must immediately follow the FILE statement.

```
DIV      1
        DEPT   3
        EMPNO  5
        NAME   20
        FILLER 4
        HOURS  3  NU   1  E
```

Note: The system assumes that the beginning location of the first field definition is position 1 in the record.

The data type for DIV is character. The data type default is represented by the keyword CH. For example, the data definition statement for DIV is as follows:

```
DIV 1 CH
DEPT 3
```

Because the first field definition defines DIV as 1 byte long (beginning with position 1), the beginning location for DEPT is position 2.

A field definition between NAME and HOURS is required for the report. A data name could have been used; however, the reserved word FILLER was used instead.

```
FILLER 4
```

A reserved word is defined as belonging to the VISION:Results program. For example, you can use the reserved word FILLER, but only in field definitions. FILLER can be used many times when defining fields, but it can never be used in procedure logic. Some other reserved words can be used in both field definitions and procedure logic. All commands, keywords, and special names (like FILLER) have rules that govern them. They can be used only when the rules are followed. For a complete list of reserved words, see [Appendix A: Reserved Words](#).

The default data type is CH for character data. When a field is defined or defaults to character data, any value is acceptable. HOURS is a numeric (NU) field. Only values 0 through 9 are valid. The numeric value following the data type defines the number of digits to the right of an assumed decimal point. The edit code E provides suppression of leading zeroes in the printed value. For example, the value 055 prints as 5.5, the value 002 prints as .2, and the value 205 prints as 20.5. For complete details, see *Advantage VISION:Results for z/OS Getting Started*.

```
HOURS 3 NU 1 E
```

Note: Only the first 36 bytes of the record are defined, not the entire 80. The remaining fields can be defined using data names, or FILLER with a length of 44 as shown in the following example.

DIV	1		DIV	1
DEPT	3		DEPT	3
EMPNO	5		EMPNO	5
NAME	20		NAME	20
FILLER	4		FILLER	4
HOURS	3 NU 1 E	OR	HOURS	3 NU 1 E
SUPERVISOR	20		FILLER	44
HIREDATE	6 NU D			
PHONE	7 NU			
SOC_SEC	9 S			
COSCTR	2			

LIST Statement

The LIST statement defines the actual report—*what* should be printed and *where* it prints.

The LIST statement is the only statement specified in the procedure logic. Therefore, because VISION:Results automatically reads the file, EMPMSTR, all records are reported on.

```
LIST DIV DEPT EMPNO NAME HOURS
```

The first field that prints (from left to right) on the report is the contents of the field called DIV, followed by DEPT, EMPNO, and so on. The contents of the field HOURS are edited (zero suppression, decimal insertion) prior to being printed.

```

01/15/01
DIV  DEPT  EMPNO  NAME  HOURS  PAGE  1
1    103   10692  W A ALTMAN  40.0
2    108   02289  S ALTENBERG 39.5
2    126   03007  C R SMITH   40.0
1    103   12352  B B CROTHERS 37.0
1    103   13126  J R LAWRY   32.0
1    209   00178  J WELL      40.0
1    103   11967  R SPRUCE    9.5
2    108   01273  W P MATHERS 38.0
2    108   02413  M R ROSEN   31.0
2    108   03523  L A MARTIN  33.0
3    110   00121  J NEWMAN    44.5
2    126   03106  B BLAKE     40.0
2    127   03132  F A CALLEY  35.5
3    106   04142  P R JONES   40.0
3    106   23415  S L HALPER  31.5
3    110   87214  S KRANZ     48.0
3    110   30241  P LING      51.0

630.5  GRAND TOTAL
17 RECORDS LISTED

```

Figure 94 Report Generated Using USERDEFAULT Mode

- The current date in mm/dd/yy format is automatically generated when using the USERDEFAULT option, unless the DYLINSTL parameter DYL4YEAR is in effect. In that case, the current date is in mm/dd/yyyy format instead.
- The current page number is printed and maintained automatically when using the USERDEFAULT option.
- The EMPMSTR file is not in any particular sequence.
- Edit code E forces leading zero suppression and decimal point insertion. If the E edit code is not used on the HOURS field definition, the leading zero appears in front of the first significant digit (where a significant digit is 1 through 9).
- VISION:Results automatically totals and prints the contents of each record's HOURS field and prints the literal GRAND TOTAL to the right of the accumulated hours. All non-character fields are totaled automatically, unless otherwise specified, when using the USERDEFAULT option.
- When using the USERDEFAULT option, an automatic count of all records prints on the report.

SORT Statement

The SORT statement consists of the SORT command followed by the file name telling VISION:Results where to find the record to be sorted. Following the file name is the keyword USING, which always precedes the names of the fields you are sorting on.

In the program, all records are sorted before they are printed. VISION:Results automatically reads a record and releases it. A second record is read, released to the sort, and so on, until there are no records left on the file. When it reaches the end of the file, VISION:Results automatically sequences (sorts) all of the records on the DIV field. Then it releases one record at a time to the statements immediately following the SORT statement. In this case, it is the LIST statement.

You can SORT sequential, DAM, BDAM, [ISAM](#), or VSAM files. Entire records or portions of records can be sorted at any point in your program.

```
SORT      EMPMSTR      USING      DIV
```

USING is a keyword identifying that data names follow. DIV in this position of the SORT statement means that the field is to be sorted in ascending order.

```
01/15/01
```

DIV	DEPT	EMPNO	NAME	HOURS	PAGE	1
1	103	10692	W A ALTMAN	40.0		
1	103	11967	R SPRUCE	9.5		
1	209	00178	J WELL	40.0		
1	103	13126	J R LAWRY	32.0		
1	103	12352	B B CRUTHERS	37.0		
2	126	03007	C R SMITH	40.0		
2	108	01273	W P MATHERS	38.0		
2	108	03523	L A MARTIN	33.0		
2	108	02413	M R ROSEN	31.0		
2	108	02289	S ALTENBERG	39.5		
2	127	03132	F A CALLEY	35.5		
2	126	03106	B BLAKE	40.0		
3	110	00121	J NEWMAN	44.5		
3	106	04142	P R JONES	40.0		
3	110	30241	P LING	51.0		
3	110	87214	S KRANZ	48.0		
3	106	23415	S L HALPER	31.5		
				630.5	GRAND TOTAL	
						17 RECORDS LISTED

Figure 95 Sorted Report

- The report is now in division (DIV) number sequence. Division 1 appears before division 2, and division 2 appears before division 3.
- Department (DEPT) numbers are not in sequence within their respective divisions (DIV).

Example 2 Using the SORT Statement

The report generated from the EMPMSTR file in the previous section is not in any particular sequence. In order to print in division number sequence, add the SORT statement to the procedure logic of the program. The SORT statement gives you the ability within your program to sort data before it is reported on or written.

```
OPTION USERDEFAULT
FILE EMPMSTR FB 80 800
    DIV      1
    DEPT     3
    EMPNO    5
```

```
NAME 20
FILLER 4
HOURS 3 NU 1 E
SORT EMPMSTR USING DIV
LIST DIV DEPT EMPNO NAME HOURS
```

CONTROL Statement

The CONTROL statement defines subtotals or automatic spacing by specifying the data names of the field definitions involved.

A data name must follow a CONTROL statement.

When using the USERDEFAULT option, you can place the CONTROL statement anywhere within the procedure logic.

A subtotal (more commonly called a control break) occurs whenever VISION:Results recognizes a change in the value (contents) of the data names specified in the CONTROL statement. In the program, the data name is DIV.

```
CONTROL DIV
```

The USERDEFAULT option automatically totals and prints all non-character fields if either of the following occurs:

- A CONTROL statement is specified in the program.
- The LIST statement contains the data name of a field that is defined as non-character.

All of the field definitions in the example are character (CH) by default, except the HOURS field, which is defined as numeric (NU).

```
01/15/01
```

DIV	DEPT	EMPNO	NAME	HOURS	PAGE	1
1	103	13126	J R LAWRY	32.0		
1	103	12352	B B CRUTHERS	37.0		
1	103	11967	R SPRUCE	9.5		
1	103	10692	W A ALTMAN	40.0		
1	209	00178	J WELL	40.0		
				158.5	DIV TOTAL	
2	108	01273	W P MATHERS	38.0		
2	108	02413	M R ROSEN	31.0		
2	108	02289	S ALTENBERG	39.5		
2	127	03132	F A CALLEY	35.5		
2	126	03106	B BLAKE	40.0		
2	126	03007	C R SMITH	40.0		
2	108	03523	L A MARTIN	33.0		
				237.0	DIV TOTAL	
3	110	00121	J NEWMAN	44.5		
3	110	30241	P LING	51.0		

Figure 96 Report Generated Using the CONTROL Statement (Page 1 of 2)

3	110	87214	S	KRANZ	48.0	
3	106	23415	S	L HALPER	31.5	
3	106	04142	P	R JONES	40.0	
					215.0	DIV TOTAL
					630.5	GRAND TOTAL
						17 RECORDS LISTED

Figure 96 Report Generated Using the CONTROL Statement (Page 2 of 2)

- The department number (DEPT) within its respective division (DIV) is not in sequence.
- The CONTROL DIV information generated the accumulated total by division and the literal DIV TOTAL automatically. Note that only the HOURS field is accumulated.

Example 3 Using the CONTROL Statement

In this example, the total number of hours worked within each division is added to the report. The CONTROL statement accomplishes this task.

```

OPTION USERDEFAULT
FILE EMPMSTR FB 80 800
    DIV      1
    DEPT     3
    EMPNO    5
    NAME     20
    FILLER   4
    HOURS    3  NU  1  E
CONTROL DIV
SORT EMPMSTR USING DIV
LIST DIV DEPT EMPNO NAME HOURS

```

CONTROL Statement on Multiple Fields

When specifying more than one data name in a CONTROL statement, the order in which they are positioned is important. Remember that this statement is used to produce control breaks, where subtotals or spacing is handled automatically.

The first data name specified in the CONTROL statement causes the minor control break, and the last data name specifies the major control break. When any control break occurs, all lower-level control breaks also occur.

Up to six data names are allowed in the CONTROL statement.

In the example, DEPT is the minor control break and DIV is the major control break. All departments within their respective divisions are to be in sequence.

Example 4 Using the CONTROL Statement on Multiple Fields

The previous report is not quite complete. In this example, to be certain that the proper totals are taken, the file is sorted by department (DEPT) within division (DIV). The report also shows the accumulated hours (HOURS) by department within division, as well as reporting on the cumulative hours by division.

To accomplish this, add the data name DEPT to both the CONTROL and SORT statements.

```
OPTION USERDEFAULT
FILE EMPMSTR FB 80 800
    DIV      1
    DEPT     3
    EMPNO    5
    NAME     20
    FILLER   4
    HOURS    3   NU   1   E
CONTROL DEPT DIV
SORT EMPMSTR USING DIV DEPT
LIST DIV DEPT EMPNO NAME HOURS
```

Placement of Data Names on CONTROL and SORT Statements

The placement order of the data names specified on the SORT statement is important. The SORT statement specification of data names is the reverse of the CONTROL statement. The major field is sorted first, followed by the minor fields. The number of fields that can be sorted is unlimited.

In the example, DEPT is to be sorted within DIV.

01/15/01	DIV	DEPT	EMPNO	NAME	HOURS	PAGE	1
	1	103	11967	R SPRUCE	9.5		
	1	103	10692	W A ALTMAN	40.0		
	1	103	13126	J R LAWRY	32.0		
	1	103	12352	B B CRUTHERS	37.0		
					118.5	DEPT TOTAL	
	1	209	00178	J WELL	40.0		
					40.0	DEPT TOTAL	
					158.5	DIV TOTAL	
	2	108	01273	W P MATHERS	38.0		
	2	108	03523	L A MARTIN	33.0		
	2	108	02413	M R ROSEN	31.0		
	2	108	02289	S ALTENBERG	39.5		
					141.5	DEPT TOTAL	
	2	126	03007	C R SMITH	40.0		
	2	126	03106	B BLAKE	40.0		
					80.0	DEPT TOTAL	
	2	127	03132	F A CALLEY	35.5		
					35.5	DEPT TOTAL	

Figure 97 Report Sorted and Subtotaled on Multiple Fields (Page 1 of 2)

```

257.0  DIV TOTAL
      3    106    23415  S L HALPER    31.5
      3    106    04142  P R JONES    40.0
                                     71.5  DEPT TOTAL
      3    110    00121  J  NEWMAN    44.5
      3    110    30241  P   LING    51.0
      3    110    87214  S   KRANZ    48.0
                                     143.5  DEPT TOTAL
                                     215.0  DIV TOTAL
                                     630.5  GRAND TOTAL
                                     17 RECORDS LISTED

```

Figure 97 Report Sorted and Subtotaled on Multiple Fields (Page 2 of 2)

- The SORT statement causes the department number (DEPT) to be in sequence within each division (DIV).
- The CONTROL DEPT DIV generated the totals for DEPT within DIV and for DIV itself. If the CONTROL statement is not used, only the grand total of HOURS is printed.

Suppress Keyword

The SUPPRESS keyword is added to the LIST statement. When SUPPRESS precedes a data name, VISION:Results prints the contents of this field only once, unless the value of the field changes. A change in the value of a field specified on a CONTROL statement also causes this suppressed field to print again.

Do not abbreviate the keyword SUPPRESS.

```
LIST SUPPRESS DIV DEPT EMPNO NAME HOURS
```

Example 5 Using the SUPPRESS Keyword

The previous report is complete; however, you might want to make it easier to read. You can accomplish this by suppressing the printing of redundant information (for example, the division number).

```

OPTION USERDEFAULT
FILE EMPMSTR FB 80 800
    DIV      1
    DEPT     3
    EMPNO    5
    NAME     20
    FILLER   4
    HOURS    3  NU  1  E
CONTROL DEPT DIV
SORT EMPMSTR USING DIV DEPT
LIST SUPPRESS (DIV DEPT) EMPNO NAME HOURS

```

Example 6 Using the SUPPRESS Keyword on Multiple Fields

The SUPPRESS keyword can also be used on multiple fields. Depending on the order of the fields you want to suppress, you can use either one of the following methods.

For example, to suppress printing of the DIV and DEPT fields, use the keyword directly before each field:

```
LIST SUPPRESS DIV SUPPRESS DEPT EMPNO NAME HOURS
```

Or, you can use parentheses to suppress printing of fields, provided the fields follow one another in the LIST statement. Because DIV and DEPT follow one another (left to right) on the report, you can shorten the coding as follows:

```
LIST SUPPRESS (DIV DEPT) EMPNO NAME HOURS
```

```
01/15/01
```

DIV	DEPT	EMPNO	NAME	HOURS	PAGE	1
1	103	11967	R SPRUCE	9.5		
		10692	W A ALTMAN	40.0		
		13126	J R LAWRY	32.0		
		12352	B B CRUTHERS	37.0		
				118.5	DEPT TOTAL	
1	209	00178	J WELL	40.0		
				40.0	DEPT TOTAL	
				158.5	DIV TOTAL	
2	108	01273	W P MATHERS	38.0		
		03523	L A MARTIN	33.0		
		02413	M R ROSEN	31.0		
		02289	S ALTENBERG	39.5		
				141.5	DEPT TOTAL	
2	126	03007	C R SMITH	40.0		
		03106	B BLAKE	40.0		
				80.0	DEPT TOTAL	
2	127	03132	F A CALLEY	35.5		
				35.5	DEPT TOTAL	
				257.0	DIV TOTAL	
3	106	23415	S L HALPER	31.5		
		04142	P R JONES	40.0		
				71.5	DEPT TOTAL	
3	110	00121	J NEWMAN	44.5		
		30241	P LING	51.0		
		87214	S KRANZ	48.0		
				143.5	DEPT TOTAL	
				215.0	DIV TOTAL	
				630.5	GRAND TOTAL	

17 RECORDS LISTED

Figure 98 Report Using the SUPPRESS Keyword on Multiple Fields

- Use the keyword **SUPPRESS** to make the report more readable.
- Both **DIV** and **DEPT** are suppressed when repeated and printed whenever a change of value occurs in either **DIV** or **DEPT**.

Field Definition Statement

Unless otherwise specified, the column heading for any field is its data name. For example, the data name **DIV** had a column heading of **DIV** in the previous examples. Column heading overrides are denoted by parentheses **()**.

```
DIV      1      (DIVISION)
```

To override column headings with a two or more line heading, indicate line breaks by using a single quotation mark **('')**:

```
EMPNO   5      (EMPLOYEE'NUMBER)
```

The heading will be placed and centered as follows:

```
EMPLOYEE  
NUMBER
```

The maximum length of an overriding column heading is 30 characters between parentheses, including any single quotation marks **('')**. Up to nine column headings are allowed.

```
01/15/01
```

	DIVISION	DEPT	EMPLOYEE NUMBER	NAME	HOURS	PAGE	1
	1	103	11967	R SPRUCE	9.5		
			10692	W A ALTMAN	40.0		
			13126	J R LAWRY	32.0		
			12352	B B CRUTHERS	37.0		
					118.5	DEPT TOTAL	
	1	209	00178	J WELL	40.0		
					40.0	DEPT TOTAL	
					158.5	DIVISION TOTAL	
	2	108	01273	W P MATHERS	38.0		
			03523	L A MARTIN	33.0		
			02413	M R ROSEN	31.0		
			02289	S ALTENBERG	39.5		
					141.5	DEPT TOTAL	
	2	126	03007	C R SMITH	40.0		
			03106	B BLAKE	40.0		
					80.0	DEPT TOTAL	
	2	127	03132	F A CALLEY	35.5		
					35.5	DEPT TOTAL	

Figure 99 Report with Column Headings Overriden (Page 1 of 2)

					257.0	DIVISION TOTAL
3	106	23415	S L	HALPER	31.5	
		04142	P R	JONES	40.0	
					71.5	DEPT TOTAL
3	110	00121	J	NEWMAN	44.5	
		30241	P	LING	51.0	
		87214	S	KRANZ	48.0	
					143.5	DEPT TOTAL
					215.0	DIVISION TOTAL
					630.5	GRAND TOTAL
						17 RECORDS LISTED

Figure 99 Report with Column Headings Overridden (Page 2 of 2)

- Column heading overrides are specified in DIV and EMPNO field definitions. The data name default is used on all other fields printed.
- The literal for division total is now DIVISION TOTAL rather than DIV TOTAL. VISION:Results automatically uses the column heading specified or defaults to the data name if used in a CONTROL statement.

Example 7 Overriding Default Column Headings

Rather than default to the data name as a column heading for the division and department fields, provide column headings for DIV and EMPNO. For DIV, use the column heading DIVISION and for EMPNO, use the column heading EMPLOYEE NUMBER (place EMPLOYEE on top of NUMBER).

This is done by using the field definition statements.

```
OPTION USERDEFAULT
FILE EMPMSTR FB 80 800
  DIV      1      (DIVISION)
  DEPT     3
  EMPNO    5      (EMPLOYEE'NUMBER)
  NAME     20
  FILLER   4
  HOURS    3      NU    1    E
CONTROL DEPT DIV
SORT EMPMSTR USING DIV DEPT
LIST SUPPRESS (DIV DEPT) EMPNO NAME HOURS
```

Title Statements

Use title (T1) statements to enter the report title. In the USERDEFAULT mode, up to eight title lines can be coded. They are automatically centered unless you specify otherwise. Variable information, such as date, page number, and user data, can be placed in title statements.

T1 'HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION'
WITH 2 AFTER

A maximum of eight title statements can be used in USERDEFAULT mode. This is because one of the nine lines normally available in the standard mode is automatically used for placement of the date and time on the report.

Figure 100 Adding a Title to a Report

The title, as specified with the T1 statement, is centered automatically on the report.

Example 8 Using Title Statements

Add the title HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION to the report. This requires a title statement.

```
1  OPTION USERDEFAULT
2  FILE EMPMSTR FB 80 800
3      DIV      1      (DIVISION)
4      DEPT     3
5      EMPNO    5      (EMPLOYEE'NUMBER)
6      NAME     20
7      FILLER   4
8      HOURS    3  NU   1   E
9  CONTROL DEPT DIV
10 SORT EMPMSTR USING DIV DEPT
11 LIST SUPPRESS (DIV DEPT) EMPNO NAME HOURS
12 T1 'HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION'
13     WITH 2 AFTER
```

IF Statement

Only those records are selected in which DIV is 2. Any records that do not meet the criteria are rejected; they are not sorted, reported on, or written to the output file.

EQ is the keyword that identifies the relational operator equal to.

The keyword NEXT in the statement means that if the compare is true (DIV equals 2), the next statement is to be executed. NEXT is a noise word and is never required. Note, however, that the NEXT keyword uses extra code.

Note: Rejected records are not sorted, nor do they appear on the report.

The following statement has an implied NEXT keyword immediately following the 2 and preceding the ELSE.

```
IF DIV EQ 2 ELSE REJECT ENDIF
```

The keyword ELSE signals to VISION:Results that a false condition directive immediately follows. REJECT is a keyword indicating that if a false condition occurs, VISION:Results rejects this record and gets another.

The ENDIF keyword is always required to terminate an IF statement.

The following are valid relational operators you can code in IF statements:

EQ	Equal to
NE	Not equal to
LT	Less than

LE Less than or equal to
 GT Greater than
 GE Greater than or equal to

```

01/15/01                                     PAGE      1
                                     HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION
DIVISION  DEPT  EMPLOYEE  NAME  HOURS
          NUMBER
          2      108    02289    S  ALTENBERG    39.5
                   01273    W P MATHERS    38.0
                   03523    L A MARTIN    33.0
                   02413    M R ROSEN    31.0
                                     141.5  DEPT TOTAL
          2      126    03106    B  BLAKE    40.0
                   03007    C R SMITH    40.0
                                     80.0  DEPT TOTAL
          2      127    03132    F A CALLEY    35.5
                                     35.5  DEPT TOTAL
                                     257.0  DIVISION TOTAL
                                     257.0  GRAND TOTAL
                                     7 RECORDS LISTED

```

Figure 101 IF Statement Used in Generating a Report

Only division 2 is selected and printed on the report.

Example 9 Using an IF Statement

An IF statement allows you to select data or certain records based on almost any criteria. It is needed in those cases where you want to process only records of a certain type, or you need to execute logic depending on conditions.

Using an IF statement, select and sort only those records from the employee master file, EMPMSTR, whose division number is 2.

```

OPTION USERDEFAULT
FILE EMPMSTR FB 80 800
  DIV      1      (DIVISION)
  DEPT     3
  EMPNO    5      (EMPLOYEE'NUMBER)
  NAME     20
  FILLER   4
  HOURS    3      NU    1    E
CONTROL DEPT DIV
IF DIV EQ 2 NEXT ELSE REJECT ENDIF
SORT EMPMSTR USING DIV DEPT
LIST SUPPRESS (DIV DEPT) EMPNO NAME HOURS
T1 'HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION'
  WITH 2 AFTER

```

IF Statement with Implied OR

This format of the IF statement implies that an OR is present. When expanded, the IF says, "If the record read has a division field value of 2 or 3, continue processing (sort the record); otherwise, ignore (reject) the record."

You can also write the IF statement as follows:

```
IF DIV EQ 2 OR DIV EQ 3 NEXT ELSE REJECT ENDIF
```

```
01/15/01                                     PAGE      1
                                     HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION
DIVISION  DEPT  EMPLOYEE      NAME      HOURS
              NUMBER
      2      108  02289      S  ALTENBERG      39.5
                  01273      W P MATHERS      38.0
                  03523      L A MARTIN      33.0
                  02413      M R ROSEN      31.0
                                     141.5  DEPT TOTAL
      2      126  03106      B  BLAKE      40.0
                  03007      C B SMITH      40.0
                                     80.0  DEPT TOTAL
      2      127  03132      F A CALLEY      33.5
                                     33.5  DEPT TOTAL
                                     257.0  DIVISION TOTAL
      3      106  04142      P B JONES      40.0
                  23415      S L HALPER      31.5
                                     71.5  DEPT TOTAL
      3      110  87214      S  KRANZ      48.0
                  00121      J  NEWMAN      44.5
                  30241      P  LING      51.0
                                     143.5  DEPT TOTAL
                                     215.0  DIVISION TOTAL
                                     472.0  GRAND TOTAL
                                     12 RECORDS LISTED
```

Figure 102 Report Generated with an IF-OR Statement

Divisions 2 and 3 are the only divisions selected, sorted, and reported on.

Example 10 Using an IF (IF-OR) Statement

Select and sort only those records from the employee master file, EMPMSTR, with a division number of 2 or 3.

To do this, add additional code to the existing IF statement.

```
OPTION USERDEFAULT
FILE EMPMSTR FB 80 800
  DIV      1      (DIVISION)
  DEPT     3
  EMPNO    5      (EMPLOYEE'NUMBER)
  NAME     20
  FILLER   4
  HOURS    3      NU    1    E
```

```
CONTROL DEPT DIV
IF DIV EQ 2 3 NEXT ELSE REJECT ENDIF
SORT EMPMSTR USING DIV DEPT
LIST SUPPRESS (DIV DEPT) EMPNO NAME HOURS
T1 'HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION'
  WITH 2 AFTER
```

IF Statement Using THRU

These IF statements have the same meaning:

```
IF DIV EQ 2 THRU 3 NEXT ELSE REJECT ENDIF
IF DIV EQ 2 3 NEXT ELSE REJECT ENDIF
```

2 THRU 3 tells VISION:Results to select all divisions between 2 and 3, inclusive.

Using the THRU keyword, you can shorten the following statement as indicated:

```
IF DIV EQ 2 3 4 5 6 7 8 NEXT ELSE REJECT ENDIF
IF DIV EQ 2 THRU 8 NEXT ELSE REJECT ENDIF
```

01/15/01	HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION					PAGE	1
DIVISION	DEPT	EMPLOYEE NUMBER	NAME	HOURS			
2	108	02289	S ALTENBERG	39.5			
		01273	W P MATHERS	38.0			
		03523	L A MARTIN	33.0			
		02413	M R ROSEN	31.0			
				141.5	DEPT TOTAL		
2	126	03106	B BLAKE	40.0			
		03007	C B SMITH	40.0			
				80.0	DEPT TOTAL		
2	127	03132	F A CALLEY	33.5			
						33.5	DEPT TOTAL
				257.0	DIVISION TOTAL		
3	106	04142	P B JONES	40.0			
		23415	S L HALPER	31.5			
				71.5	DEPT TOTAL		
3	110	87214	S KRANZ	48.0			
		00121	J NEWMAN	44.5			
		30241	P LING	51.0			
				143.5	DEPT TOTAL		
				215.0	DIVISION TOTAL		
				472.0	GRAND TOTAL		
12 RECORDS LISTED							

Figure 103 Report Generated with a THRU Statement

Divisions 2 through 3 are selected, sorted, and reported on.

Example 11 Using the THRU Keyword in an IF Statement

Select and sort only those records from the employee master file, EMPMSTR, whose division number is 2 or 3. In this example, use the keyword THRU (or THROUGH) in the IF statement to select divisions 2 and 3.

```
OPTION USERDEFAULT
FILE EMPMSTR FB 80 800
    DIV      1      (DIVISION)
    DEPT     3
    EMPNO    5      (EMPLOYEE'NUMBER)
    NAME     20
    FILLER   4
    HOURS    3      NU    1    E
CONTROL DEPT DIV
IF DIV EQ 2 THRU 3 NEXT ELSE REJECT ENDIF
SORT EMPMSTR USING DIV DEPT
LIST SUPPRESS (DIV DEPT) EMPNO NAME HOURS
T1 'HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION'
    WITH 2 AFTER
```

IF Statement with ELSE NEXT ENDIF

You can also state the 40 hours in the IF statement as:

```
IF HOURS GE 40 GOTO SORTREC ELSE NEXT ENDIF
```

This provides a second chance for the record to be selected by division (DIV) number.

SORTREC as a Tag Name

Without the colon, SORTREC is a tag name. Tag names are generally used to branch from one statement to another in procedure logic. When used in conjunction with a GOTO, it is specified as:

```
GOTO SORTREC
```

Tag names can be from 1 through 10 characters long. They may be alphabetic, numeric, or a combination of both. For example:

```
100              REC100
A102             200REC
```

SORTREC: as a Reference Point

With the colon, SORTREC: is a reference point.

```
SORTREC:
```

All reference points must have the colon (:) appended to the rightmost character. Thus, a 10-character tag name has a reference point length of 11.

```
ABCDEFGH IJ:
0123456789:
```

01/15/01

PAGE 1

HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION

DIVISION	DEPT	EMPLOYEE NUMBER	NAME	HOURS	
1	103	10692	W A ALTMAN	40.0	
				40.0	DEPT TOTAL
1	209	00178	J WELL	40.0	
				40.0	DEPT TOTAL
				80.0	DIVISION TOTAL
2	108	03523	L A MARTIN	33.0	
		02413	M R ROSEN	31.0	
		02289	S ALTENBERG	39.5	
		01273	W P MATHERS	38.0	
				141.5	DEPT TOTAL
2	126	03106	B BLAKE	40.0	
		03007	C R SMITH	40.0	
				80.0	DEPT TOTAL
2	127	03132	F A CALLEY	35.5	
				35.5	DEPT TOTAL
				257.0	DIVISION TOTAL
3	106	04142	P R JONES	40.0	
		23415	S L HALPER	31.5	
				71.5	DEPT TOTAL
3	110	87214	S KRANZ	48.0	
		00121	J NEWMAN	44.5	
		30241	P LING	51.0	
				143.5	DEPT TOTAL
				215.0	DIVISION TOTAL
				552.0	GRAND TOTAL

14 RECORDS LISTED

Figure 104 Using SORTREC as a Reference Point

The employees whose hours were greater than or equal to 40 now appear in the report. Two employees from division 1 were selected along with those employees who worked in divisions 2 and 3.

Example 12 Using SORTREC as a Tag Name and as a Reference Point

Select all employees who worked 40 hours or more, regardless of whether they work in division 2 or 3. Also, select all employees who work in division 2 and division 3.

```

OPTION USERDEFAULT
FILE EMPMSTR FB 80 800
    DIV      1      (DIVISION)
    DEPT     3
    EMPNO    5      (EMPLOYEE'NUMBER)
    NAME     20
    FILLER   4
    HOURS    3      NU    1    E
CONTROL DEPT DIV
IF HOURS GE 40.0 GOTO SORTREC ELSE NEXT REJECT ENDIF
IF DIV EQ 2 THRU 3 NEXT ELSE REJECT ENDIF

```

```

SORTREC:
  SORT EMPMSTR USING DIV DEPT
  LIST SUPPRESS (DIV DEPT) EMPNO NAME HOURS
T1 'HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION'
  WITH 2 AFTER

```

COPY Statement

Use the COPY statement when you code and catalog the file or field definitions and want to include them as part of your program. This is typically done when a particular file is used again and again to produce reports.

The COPY statement is not limited to file and field definitions. You can also use it to copy in procedure logic. For more information about the COPY commands, see [Chapter 15: Using the COPY Commands](#).

```

      OPTION USERDEFAULT
      COPY EMPMSTR
COPIED FILE EMPMSTR FB      80  800
COPIED      DIV              1
COPIED      DEPT             3
COPIED      EMPNO            5
COPIED      NAME             20
COPIED      FILLER            4
COPIED      HOURS             3  NU    1    E
COPIED      SUPERVISOR       20
COPIED      HIREDATE          6  NU          D
COPIED      FILLER           18
CONTROL DEPT DIV
IF HOURS GE 40.0 GOTO SORTREC ELSE NEXT ENDIF
IF DIV EQ 2 THRU 3 NEXT ELSE REJECT ENDIF
SORTREC:
  SORT EMPMSTR USING DIV DEPT
  LIST SUPPRESS (DIV DEPT) EMPNO NAME HOURS
T1 'HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION' WITH 2 AFTER

```

Figure 105 Source Listing Showing File and Field Definitions Copied

Contents of the member name/book name EMPMSTR are identified in the source listing by the word COPIED.

```

01/15/01
                                HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION
                                PAGE 1

```

DIVISION	DEPT	EMPLOYEE NUMBER	NAME	HOURS	
1	103	10692	W A ALTMAN	40.0	
				40.0	DEPT TOTAL
1	209	00178	J WELL	40.0	
				40.0	DEPT TOTAL
				80.0	DIVISION TOTAL
2	108	03523	L A MARTIN	33.0	
		02413	M R ROSEN	31.0	
		02289	S ALTENBERG	39.5	
		01273	W P MATHERS	38.0	
				141.5	DEPT TOTAL
2	126	03106	B BLAKE	40.0	
		03007	C R SMITH	40.0	

Figure 106 Report Generated Using Source Listing (Page 1 of 2)

				80.0	DEPT TOTAL
2	127	03132	F A CALLEY	35.5	
				35.5	DEPT TOTAL
				257.0	DIVISION TOTAL
3	106	04142	P R JONES	40.0	
		23415	S L HALPER	31.5	
				71.5	DEPT TOTAL
3	110	87214	S KRAVZ	48.0	
		00121	J NEWMAN	44.5	
		30241	P LING	51.0	
				143.5	DEPT TOTAL
				215.0	DIVISION TOTAL
				552.0	GRAND TOTAL
					14 RECORDS LISTED

Figure 106 Report Generated Using Source Listing (Page 2 of 2)

Example 13 Using the COPY Statement

In this example, the file and field definitions have been cataloged in a source library, and the employee master file and field definitions have been given the name of EMPMSTR. Because of this, you do not need to define the file or field definitions again. To retrieve the definitions, you need only supply a COPY statement in your program. This example is identical to the last example.

```

OPTION USERDEFAULT
COPY EMPMSTR
CONTROL DEPT DIV
IF HOURS GE 40.0 GOTO SORTREC ELSE NEXT REJECT ENDIF
IF DIV EQ 2 THRU 3 NEXT ELSE REJECT ENDIF
SORTREC:
  SORT EMPMSTR USING DIV DEPT
  LIST SUPPRESS (DIV DEPT) EMPNO NAME HOURS
T1 'HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION'
  WITH 2 AFTER

```

Example 14 Using COBOL in the COPY Statement

In this example, COBOL definitions are maintained for the employee master file. Instead of defining the field definitions with VISION:Results, you can use the predefined COBOL data definitions. The FILE statement needs to be supplied, but all field definitions come from the member name/book name, EMPMSTCO.

```

OPTION USERDEFAULT
FILE EMPMSTR FB 80 800
COPY EMPMSTCO COBOL
CONTROL DEPT DIV
IF HOURS GE 40.0 GOTO SORTREC ELSE NEXT REJECT ENDIF
IF DIV EQ 2 THRU 3 NEXT ELSE REJECT ENDIF
SORTREC:
  SORT EMPMSTR USING DIV DEPT
  LIST SUPPRESS (DIV DEPT) EMPNO NAME HOURS
T1 'HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION'
  WITH 2 AFTER

```

```

OPTION USERDEFAULT
FILE EMPMSTR FB      80 800
COPY EMPMSTCO        COBOL
WARNING DYL-188W  HYPHENS (-) IN COPIED DATANAMES WILL BE CHANGED INTERNALLY TO UNDERSCORES ( _ )
COPIED      01 EMPLOYEE-RECORD.
COPIED      05 DIV          PIC X.
COPIED      05 DEPT        PIC XXX.
COPIED      05 EMPNO       PIC X(5).
COPIED      05 NAME        PIC X(20).
COPIED      05 FILLER      PIC X(4).
COPIED      05 HOURS       PIC 9(6).
COPIED      05 SUPERVISOR  PIC X(20).
COPIED      05 HIREDATE    PIC 9(6).
COPIED      05 FILLER      PIC X(18).
CONTROL DEPT      DIV
IF HOURS GE 40.0 GOTO SORTREC ELSE NEXT ENDIF
IF DIV EQ 2 THRU 3 NEXT ELSE REJECT ENDIF
SORTREC:
  SORT EMPMSTR USING DIV DEPT
  LIST SUPPRESS (DIV DEPT) EMPNO NAME HOURS
  T1 'HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION' WITH 2 AFTER

```

Figure 107 Source Library with COBOL

The contents of the member name/book name EMPMSTCO are identified in the source listing by the word COPIED.

```

01/15/01
                                HOURLY EMPLOYEES BY DEPARTMENT WITHIN DIVISION
                                PAGE 1

```

DIVISION	DEPT	EMPLOYEE NUMBER	NAME	HOURS	
1	103	10692	W A ALTMAN	40.0	
				40.0	DEPT TOTAL
1	209	00178	J WELL	40.0	
				40.0	DEPT TOTAL
				80.0	DIVISION TOTAL
2	108	03523	L A MARTIN	33.0	
		02413	M R ROSEN	31.0	
		02289	S ALTEBERG	39.5	
		01273	W P MATHERS	38.0	
				141.5	DEPT TOTAL
2	126	03106	B BLAKE	40.0	
		03007	C R SMITH	40.0	
				80.0	DEPT TOTAL
2	127	03132	F A CALLEY	35.5	
				35.5	DEPT TOTAL
				257.0	DIVISION TOTAL
3	106	04142	P R JONES	40.0	
		23415	S L HALPER	31.5	
				71.5	DEPT TOTAL
3	110	87214	S KRANZ	48.0	
		00121	J NEWMAN	44.5	
		30241	P LING	51.0	
				143.5	DEPT TOTAL
				215.0	DIVISION TOTAL
				552.0	GRAND TOTAL

14 RECORDS LISTED

Figure 108 Report Generated Using COBOL Definitions

The ARFILE File

The following examples use the accounts receivable file (ARFILE), a fixed blocked (FB) file. Each record is 352 bytes long, and there are 15 records per block (5280).

Details are given only when a new command or keyword is presented or the example warrants more explanation.

The fields in the following examples are defined below.

Data name	Description
NAME	Name of account holder
BALANCE	Balance due
ACCOUNT	Account code
INSTLBAL	Installment payment

Data name	Size	Location	Data Type	Decimal Places
NAME	25	85	CH (default)	
BALANCE	5	170	PD	2
ACCOUNT	2	182	CH	
INSTLBAL	6	191	PD	2

Example 15 Using the IF Statement to Select Multiple Records

Using the existing example from the ARFILE, select, sort, and report on only those records whose account code is NA, EO, or FO. All other accounts should be rejected. Add the formal report title of CONTROL LISTING REPORT BY ACCOUNT to the report.

```
OPTION USERDEFAULT
FILE ARFILE FB 352 5280
  NAME 25 85 ACCOUNT 2 182
  BALANCE 5 170 PD 2 (BALANCE'OWED)
CONTROL ACCOUNT
IF ACCOUNT EQ 'NA' 'EO' 'FO' NEXT
  ELSE REJECT ENDIF
SORT ARFILE USING ACCOUNT
LIST SUPPRESS ACCOUNT NAME BALANCE
T1 'CONTROL LISTING REPORT BY ACCOUNT' WITH 2 AFTER
```

```

01/15/01
CONTROL LISTING REPORT BY ACCOUNT
PAGE 1

ACCOUNT      NAME      BALANCE
              OWED

EO      SANTA FE HOSP ASSN      15.00
        S.F.MEM.HOSP.          5.00
        CHO PYUNG, SUH          32.00
              52.00      ACCOUNT TOTAL

FO      GENVARDI, G J          43.00
        HUGHES, RAY          178.70
        HILL, GARY E          3.80
        CRANO, MICHAEL S      15.00
        CHAVEZ, RAY          15.00
        SILVA, JULIAN          .00
        TODIPE, MICHAEL      45.24
              300.74      ACCOUNT TOTAL

NA      310.75
        64.96
        1.04-
        35.00
        1.32-
        3.32-
        33.60-
        2.46-
        10,085.55-
        256.12-
        11.20-
        19.54
              9,964.36-      ACCOUNT TOTAL
              9,611.62-      GRAND TOTAL

22 RECORDS LISTED

```

Figure 109 Report on Selected Records by Account Code

Example 16 Select, Sort, and Report on Specified Records

Select, sort, and report on all records with a balance due greater than \$100.00. Select account codes NA and EO THRU FO. Reject all other accounts.

```

OPTION USERDEFAULT
FILE ARFILE FB 352 5280
NAME 25 85 ACCOUNT 2 182
BALANCE 5 170 PD 2 (BALANCE'OWED)
CONTROL ACCOUNT
IF BALANCE GT 100.00 GOTO SORTREC ENDIF
IF ACCOUNT EQ 'NA' 'EO' THRU 'FO' NEXT
ELSE REJECT ENDIF
SORTREC:
SORT ARFILE USING ACCOUNT
LIST SUPPRESS ACCOUNT NAME BALANCE
T1 'CONTROL LISTING REPORT BY ACCOUNT' 2 AFTER

```

01/15/01	CONTROL LISTING REPORT BY ACCOUNT			PAGE	1
	ACCOUNT	NAME	BALANCE OWED		
	EO	SANTA FE HCSP ASSN	15.00		
		S.F.MEM.HOSP.	5.00		
		CHO PYUNG, SUH	32.00		
			52.00	ACCOUNT TOTAL	
	FO	GENVARDI, G J	43.00		
		HUGHES, RAY	178.70		
		HILL, GARY E	3.80		
		CANO, MICHAEL S	15.00		
		CHAVEZ, RAY	15.00		
		SILVA, JULIAN	.00		
		TODIPE, MICHAEL	45.24		
			300.74	ACCOUNT TOTAL	
	IO	AGUIERA, EMILIO	108.44		
		PLACIDO, ORTEGA	413.58		
		WILSON, HERMAN	5,292.24		
			6,752.85	ACCOUNT TOTAL	
	WO	TAUTRIM, RICHARD	140.00		
		KANGAROO R.ROAD CO.	352.80		
01/15/01	CONTROL LISTING REPORT BY ACCOUNT			PAGE	3
	ACCOUNT	NAME	BALANCE OWED		
	WO	ANDREWS, CHARLES	114.52		
		BADILLY, NEIL	114.23		
		NARANJO, DELIA	115.60		
		INS CO OF NO AMER	108.25		
		KANGAROO R.ROAD CO.	265.74		
		KANGAROO R.ROAD CO.	121.55		
			1,332.69	ACCOUNT TOTAL	
			15,663.18	GRAND TOTAL	
				63 RECORDS LISTED	

Figure 110 Report on Selected Records by Account Code and Balance

Example 17 Adding a Field to a Report

To the existing program and report, add the installment payment field (INSTLBAL) to the right of BALANCE. Use the column heading INSTALLMENT PAYMENT. Prior to releasing the record to sort, increase the BALANCE field by 50%.

Remember that all non-character fields are totaled automatically, and when a control break occurs, the accumulated amount prints for those fields.

```

OPTION USERDEFAULT
FILE ARFILE FB 352 5280
  NAME 25 85 ACCOUNT 2 182
  BALANCE 5 170 PD 2 (BALANCE'OWED)
  INSTLBAL 6 191 PD 2 A (INSTALLMENT'PAYMENT)
CONTROL ACCOUNT
IF BALANCE GT 100.00 GOTO SORTREC ENDIF
IF ACCOUNT EQ 'NA' 'EO' THRU 'FO' NEXT
  ELSE REJECT ENDIF
SORTREC:

```

```

BALANCE=BALANCE * 1.5
SORT ARFILE USING ACCOUNT
LIST SUPPRESS ACCOUNT NAME BALANCE INSTLBAL
T1 'CONTROL LISTING REPORT BY ACCOUNT' WITH 2 AFTER

```

01/15/01	CONTROL LISTING REPORT BY ACCOUNT			PAGE	1
ACCOUNT	NAME	BALANCE OWED	INSTALLMENT PAYMENT		
EO	SANTA FE HOSP ASSN	22.50	2,230.17		
	S.F.MEM.HOSP.	7.50	272.78		
	CHO PYUNG, SUH	48.00	261.57		
		78.00	2,764.52	ACCOUNT TOTAL	
FO	GENVARDI, G J	64.50	419.48		
	HUGHES, RAY	268.05	47.88		
	HILL, GARY E	5.70	22.38		
	CANO, MICHAEL S	22.50	195.74		
	CHAVEZ, RAY	22.50	49.39		
	SILVA, JULIAN	.00	273.99		
	TODIPE, MICHAEL	67.86	486.21		
		451.11	1,495.07	ACCOUNT TOTAL	
IO	AGUILERA, EMILIO	162.66	197.85		
	PLACIDO, ORTEGA	620.37	486.21		
01/15/01	CONTROL LISTING REPORT BY ACCOUNT			PAGE	3
ACCOUNT	NAME	BALANCE OWED	INSTALLMENT PAYMENT		
WO	ANDREWS, CHARLES	171.78	196.87		
	BADILY, NEIL	171.34	201.46		
	NARANJO, DELIA	173.40	492.96		
	INS CO OF NO AMER	162.37	137.16		
	KANGAROO R.ROAD CO.	398.61	396.27		
	KANGAROO R.ROAD CO.	182.32	359.98		
		1,999.02	2,236.37	ACCOUNT TOTAL	
		23,494.71	39,393.48	GRAND TOTAL	
				63 RECORDS LISTED	

Figure 111 Report with a Field Added

Each BALANCE amount that printed is 50% more than the amount printed in the previous example.

NOSUM Keyword

```
INSTLBAL 6 191 PD 2 A (INSTALLMENT' PAYMENT) NOSUM
```

Note: This keyword is ignored in the standard and structured modes of operation.

The NOSUM keyword has meaning only when used in USERDEFAULT mode. All non-character fields are accumulated and printed automatically when you use a CONTROL statement and the non-character field is specified on a LIST statement, unless you use a date (D) or social security (S) edit code. You can override this automatic accumulation and subtotal printing with the NOSUM keyword on the field definition statement.

01/15/01	CONTROL LISTING REPORT BY ACCOUNT			PAGE	1
ACCOUNT	NAME	BALANCE OWED	INSTALLMENT PAYMENT		
EO	SANTA FE HOSP ASSN	22.50	2,230.17		
	S.F.MEM.HOSP.	7.50	272.78		
	CHO PYUNG, SUH	48.00	261.57		
		78.00		ACCOUNT TOTAL	
FO	GENVARDI, G J	64.50	419.48		
	CANO, MICHAEL S	22.50	195.74		
	CHAVEZ, RAY	22.50	49.39		
	SILVA, JULIAN	.00	273.99		
	TODIPE, MICHAEL	67.86	486.21		
	HUGHES, RAY	268.05	47.88		
	HILL, GARY E	5.70	22.38		
		451.11		ACCOUNT TOTAL	
IO	AGUIERA, EMILIO	162.66	197.85		
	PLACIDO, ORTEGA	620.37	486.21		
01/15/01	CONTROL LISTING REPORT BY ACCOUNT			PAGE	3
ACCOUNT	NAME	BALANCE OWED	INSTALLMENT PAYMENT		
WO	KANGAROO R.ROAD CO.	398.61	396.27		
	KANGAROO R.ROAD CO.	182.32	359.98		
	KANGAROO R.ROAD CO.	529.20	295.42		
	ANDREWS, CHARLES	171.78	196.87		
	RADILY, NEIL	171.34	201.46		
	NARANJO, DELIA	173.40	492.96		
		1,999.02		ACCOUNT TOTAL	
		23,494.71		GRAND TOTAL	
				63 RECORDS LISTED	

Figure 112 Report Using NOSUM to Suppress Subtotals

The subtotal did not accumulate or print for INSTLBAL because the NOSUM keyword was used on the INSTLBAL field definition.

Example 18 Using the NOSUM Keyword in a Field Definition

Produce the same report as in the last example, but do not automatically total and print the installment payment field when the account code value changes.

Remember that all non-character fields are totaled automatically and when a control break occurs, the accumulated amount is printed for those fields. The exception is when you use the NOSUM keyword in the field definition.

```

OPTION USERDEFAULT
FILE ARFILE FB 352 5280
  NAME 25 85 ACCOUNT 2 182
  BALANCE 5 170 PD 2 (BALANCE'OWED)
  INSTLBAL 6 191 PD 2 A (INSTALLMENT'PAYMENT) NOSUM
CONTROL ACCOUNT
IF BALANCE GT 100.00 GOTO SORTREC ENDIF
IF ACCOUNT EQ 'NA' 'EO' THRU 'FO' NEXT
  ELSE REJECT ENDIF
SORTREC:
  BALANCE=BALANCE * 1.5
  SORT ARFILE USING ACCOUNT
  LIST SUPPRESS ACCOUNT NAME BALANCE INSTLBAL
T1 'CONTROL LISTING REPORT BY ACCOUNT' WITH 2 AFTER

```


Chapter 10: Using the MOVE Command

Use the MOVE command to:

- Transfer a field or group of fields from one area to another.
- Convert data from one type to another (for example, numeric to packed, binary to packed, or packed to binary).
- Decrease or increase decimal positions of numeric, packed, or binary fields.
- Initialize fields.

The MOVE statement in its simplest form looks like this:

```
MOVE datanamex TO datanamez
```

Figure 113 MOVE Statement Form Example 1

or:

```
MOVE literal TO datanamez
```

Figure 114 MOVE Statement Form Example 2

In most instances, sending and receiving fields do not have to be of the same size or data type, or have the same number of decimal positions. VISION:Results automatically handles the data conversion and decimal alignment. Sending and receiving fields with varied locations can have their addresses modified by indexing. This permits entries in tables or segments in records to be moved from one location to another.

```
MOVE DATA1 (INW) TO DATA2 (INX)
```

You can move a character field or a group of fields, varying the length of the MOVE:

```
MOVE VARFIELD LENGTH INLEN TO XFIELD
```

MOVE statements can appear almost anywhere in your program logic, as either independent statements or within IF statements.

The following program demonstrates the various aspects of data movement and conversion:

```
FILE FILEIN FB 65 650
  ICUSTNO 5 NU  ILNLEN 2 NU  INAME 30
  IAMOUNT1 7 NU 2  IAMOUNT2 7 NU 2  IAMOUNT3 7 NU 2
  IAMOUNT4 7 NU 2
FILE FILEOUT OUTPUT FROM FILEOUT FB 39 3900
  OCUSTNO 3 PD  ONAME 20
  OAMOUNT1 4 PD 2  OAMOUNT2 4 PD 2  OAMOUNT3 4 PD 2
  OAMOUNT4 4 PD 2
WORKAREA
  AMTCNT 1 PD VALUE 0

PROCEDURE:
  MOVE SPACES TO ONAME
  MOVE ICUSTNO TO OCUSTNO
  MOVE INAME LENGTH ILNLEN TO ONAME
  MOVE 0 TO INW
  MOVE 0 TO INX
LOOP:
  MOVE IAMOUNT1 (INW) TO OAMOUNT1 (INX)
  AMTCNT=AMTCNT+1
  IF AMTCNT EQ 4
    MOVE 0 TO AMTCNT
    GOTO ENDLOOP ENDIF
  INW=INW+7
  INX=INX+4
  GOTO LOOP
ENDLOOP:
  ACCEPT
```

Figure 115 Data Movement and Conversion

The program is reformatting and generating a new master file from an existing file. The customer number (ICUSTNO) is being converted from a 5-byte numeric field (zoned decimal) to a 3-byte packed decimal field. The amount (IAMOUNT1 through IAMOUNT4) fields are being converted from 7-byte, 2-decimal numeric fields (zoned decimal) to 4-byte, 2-decimal packed fields. The character field INAME has the format of last name, first name. The ILNLEN field holds the length of the last name; this length is used to move the last name portion of INAME to the output record area.

One of the records on the input file appears as:

```
1000106MARTIN, JOHN H      1234567123456712345671234567
```

When it is reformatted, it appears in hex and graphics:

```
      MARTIN
101DCDECD444444444444441357135713571357
00C419395000000000000000246C246C246C246C
```

Move Field to Field

To transfer a field from one area to another (for example, from an input record area to an output record area), define the sending and receiving fields where appropriate, giving each a data name and stating the attributes, and then, in your procedure logic, code:

```
MOVE datanamex TO datanamez
```

Figure 116 Transfer a Field

For a list of valid MOVE combinations and the rules that govern them, see [Rules for Moving Fields on page 61](#).

```
MOVE ICUSTNO TO OCUSTNO
```

You need to pay particular attention to the rules governing MOVE operations between character fields of different lengths, especially when using indexing.

Move Field to Field—Variable Length

You can move a field or group of contiguous fields to another field where the length of the move varies depending on conditions. Use one of the following formats:

MOVE datanamex LENGTH datanamey TO datanamez	datanamey is the data name of a numeric, packed, or binary field containing the value 1 to 32767, and nnnnn is a numeric literal value of 1 to 32767.
MOVE datanamex TO datanamez LENGTH datanamey	
MOVE datanamex LENGTH nnnnn TO datanamez	
MOVE datanamex TO datanamez LENGTH nnnnn	

Figure 117 Move Field to Field—Variable Length

The sending and receiving fields must be defined as character type fields. The length specified in the MOVE statement overrides what has been specified in the field definition for both the sending and receiving fields. The sending field is left-aligned in the receiving field. The receiving field is not padded with blanks.

In the programming example, the numeric field ILNLEN contains the length of the last name portion of the field INAME. The length field is used in the MOVE statement to transfer the last name to the output area:

```
MOVE INAME LENGTH ILNLEN TO ONAME
```

or:

```
MOVE INAME TO ONAME LENGTH ILNLEN
```

For example, to move six characters of the field to the output area, code:

```
MOVE INAME LENGTH 6 TO ONAME
```

or:

```
MOVE INAME TO ONAME LENGTH 6
```

The character data type requirement for both fields does not prevent you from moving fields of various data types. For example, you can move a group of packed counters using the variable move format by redefining the first byte of the group and calling it character.

If the field you are applying the LENGTH phrase to is to be indexed, an example is:

```
MOVE INAME (INW) LENGTH ILNLEN TO ONAME
```

Move Literal to Field

You can MOVE a numeric, character, or hexadecimal literal to a field.

Move Numeric Literal to a Field

A 1- to 16-digit numeric literal can be moved to a 1- to 16-byte numeric, 1- to 16-byte packed, or 1- to 4-byte binary field. The format is:

```
MOVE nnnnn TO datanamez
```

Figure 118 Move Numeric Literal to a Field

The literal can include a leading minus sign (if negative), commas, and a decimal point. The literal can have fewer decimal positions than the receiving field; decimal alignment is performed automatically. If the literal value is too large to be held in the receiving field, a validation error occurs. The receiving field can be indexed.

The following statement clears the binary index field INW to binary zeros:

```
MOVE 0 TO INW
```

Here are some other examples of numeric literal moves:

```
MOVE 0 TO BALANCE  
MOVE -1 TO FIELD1  
MOVE 1,234.55 TO BALANCE  
MOVE 1.10 TO FACTOR (INW)
```

Move Character Literal to a Field

A 1- to 70-byte character literal can be moved to a 1- to 32767-byte character field. The format is as follows:

```
MOVE 'literal' TO datanamez
```

Figure 119 Move Character Literal to a Field

The literal must be enclosed in single or double quotation marks and must fit on one line. If the receiving field is longer than the literal value, the literal is left-aligned in the receiving field and padded with blanks. The literal cannot be longer than the receiving field. The receiving field can be indexed.

```
MOVE 'WASHINGTON' TO STATE
MOVE '2113-5' TO CODE
MOVE "2113-5" TO CODE (INW)
```

Move Hexadecimal Literal to a Field

A 1- to 9-byte literal expressed in hexadecimal notation can be moved to a 1- to 32767-byte character field. The format is:

```
MOVE X'hhhh...hh' TO datanamez
```

Figure 120 Move Hexadecimal Literal to a Field

If the receiving field is longer than the literal value, the literal is left-aligned in the receiving field and padded with blanks. The literal cannot be longer than the receiving field. The receiving field can be indexed.

```
MOVE X'01020304' TO TABLE1
```

The 4-byte hexadecimal literal is to be moved to TABLE1.

```
MOVE X'FF' TO SWITCH
MOVE X'00' TO FIELD2 (INW)
```

Considerations for MOVE literal

A variable-length MOVE cannot be performed if the sending field is a literal.

For example, this is not valid:

```
MOVE ' ' TO DATA1 LENGTH INLEN
```

A 1- to 4-byte literal expressed in hexadecimal notation can be moved to the same length binary field if the binary field has zero decimals.

If the literal being moved is shorter than the receiving field and indexing is used, data overlay could occur because the receiving field length is used to determine how many blanks to use in padding. If this is a concern, MOVE the literal to an intermediate field of the correct length, then MOVE the intermediate field to the final, indexed field. To do this, use the LENGTH keyword to specify the actual literal (intermediate field) size.

```
REC_FIELD      100
LITLEN        2 BI      VALUE 5
LITHOLD        5
.
.
.
MOVE 'XXXXX' TO LITHOLD
INX=75
MOVE LITHOLD LENGTH LITLEN TO REC_FIELD(INX)
```

Move Blanks to Character Field

To clear a character field to blanks, the formats are:

```
MOVE ' ' TO datanamez
MOVE ' ' TO datanamez (index)
MOVE SPACES TO datanamez
MOVE SPACES TO datanamez (index)
MOVE SPACES TO datanamez LENGTH datanamey
MOVE SPACES TO datanamez LENGTH nnnnn (1 to 32767 bytes)
MOVE SPACES TO datanamez (index) LENGTH datanamey
MOVE SPACES TO datanamez (index) LENGTH nnnnn
```

Figure 121 Clear a Character Field to Blanks

Instead of SPACES, you can use the words SPACE, BLANK, or BLANKS. The field being cleared can be from 1 to 32767 bytes long.

In the example, the 20-byte character field ONAME is cleared by this statement:

```
MOVE SPACES TO ONAME
```

Move Field or Literal to a Subtotal Field

During logic entered after ON CHANGE IN or ON FINAL, you can MOVE to or from a totaled field:

```
ON CHANGE IN ACCOUNT
  IF SUM BALANCE LT 0
    MOVE 0 TO SUM BALANCE ENDIF
```

The command SUM preceding BALANCE indicates that the subtotal of BALANCE for that particular account is to be accessed, not the current BALANCE field.

MOVE NUMERIC

To transfer only the numeric portion of bytes from one field to another, the formats are:

```
MOVE NUMERIC datanamex TO datanamez
MOVE NUMERIC literal TO datanamez
```

Figure 122 Numerics From One Field to Another

A 1- to 256-byte field of any data type can have its numeric bits moved to the numeric bits of a 1- to 256-byte field of any data type. In other words, the low order 4 bits of each byte in the send field are placed in the low order 4 bits of the corresponding byte of the receiving field. The high order 4 bits of each byte remain unchanged. The fields must be of equal length.

```
MOVE NUMERIC DATA1 TO DATA2
```

Assume that DATA1 is a 3-byte field of numeric data type and contains the value (hex notation) C1C2C3. DATA2 is a 3-byte field that contains the value F0F0F0. After the MOVE NUMERIC operation, DATA2 contains the value F1F2F3.

The fields involved in the MOVE NUMERIC operation can be indexed:

```
MOVE NUMERIC DATA1 (INX) TO DATA2 (INY)
```

A variable-length MOVE NUMERIC is not valid.

MOVE ZONE

To transfer only the zone portion of bytes from one field to another, the formats are:

```
MOVE ZONE datanameX TO datanameZ
MOVE ZONE literal TO datanameZ
```

Figure 123 Zones From One Field to Another

A 1- to 256-byte field of any data type can have its zone bits moved to the zone bits of a 1- to 256-byte field. The high order 4 bits of each byte in the send field are placed in the high order 4 bits of the corresponding byte of the receiving field. The low order bits remain unchanged. The fields must be of equal length.

```
MOVE ZONE DATA1 TO DATA2
```

Assume that DATA1 is a 3-byte numeric field and contains the value (hex notation) F0F0F0 (that is, a positive 000). DATA2 is a 3-byte field that contains the value C1C2C3. After the MOVE ZONE operation, DATA2 contains the value F1F2F3.

The fields involved in the MOVE ZONE operation can be indexed:

```
MOVE ZONE DATA1 (INX) TO DATA2 (INW)
```

A variable-length MOVE ZONE is not valid.

MOVE with OFFSET

To move a field and place it to the left of and adjacent to the low order 4 bits of a receiving field, the formats are:

```
MOVE OFFSET datanameX TO datanameZ
MOVE OFFSET literal TO datanameZ
```

Figure 124 MOVE with OFFSET

A 1- to 16-byte field of any data type can be moved with offset to a 1- to 16-byte field of any data type. The fields do not have to be of equal sizes. If the receiving field is larger, it is filled with high order binary zeros. If the receiving field is too short to contain all bytes of the sending field, the remaining information in the sending field is ignored.

MOVE OFFSET DATA1 TO DATA2

Assume DATA1 is a 2-byte field containing the value (hex notation) of 1234. DATA2 is a 3-byte field containing the value (hex notation) of 00000C. After the MOVE OFFSET operation, DATA2 contains the value 01234C.

In the example above, the MOVE OFFSET command converts a packed unsigned value to a valid packed signed value.

The fields involved in the MOVE OFFSET operation can be indexed:

MOVE OFFSET DATA1 (INW) TO DATA2 (INX)

A variable-length MOVE OFFSET is not valid.

COMBINE BITS Using OR

To turn certain bits on in a field, here are the formats:

COMBINE	BITS	dataname _x	OR	dataname _z
COMBINE	BITS	literal	OR	dataname _z

Figure 125 Turn Bits On

You can also express an OR where the result of the OR operation is placed in a third field. The contents of the original fields remain unchanged. These are the formats:

COMBINE	BITS	dataname _w	OR	dataname _x	INTO	dataname _z
COMBINE	BITS	literal	OR	dataname _x	INTO	dataname _z

Figure 126 Combine Bits Using OR

An OR can combine a 1- to 256-byte field of any data type into a 1- to 256-byte field of any data type.

An OR operation works in this manner: a bit position in the receiving field is set to 1 if the corresponding bit position in the sending field contains a 1.

COMBINE BITS DATA1 OR DATA2

Assume DATA1 is a 3-byte field with the value (hex notation) of 00000F and DATA2 is a 3-byte field with the value of 12345C. After the OR operation, DATA2 contains the value 12345F. The two low order bits of the field have been set to ones.

COMBINE BITS X'00000F' OR DATA2

The same result occurs as in the previous example, except a hex literal sets the bits.

```
COMBINE BITS X'00000F' OR DATA2 INTO DATA3
```

In this example, the data in DATA2 is moved to DATA3 before the OR is done. DATA2 remains unaltered.

The fields involved in the OR operation can be indexed:

```
COMBINE BITS DATA1 (INW) OR DATA2 (INX)
```

A variable-length OR operation is not valid.

AND a Field or Literal with a Field

To combine one field with another using AND, use these formats:

```
COMBINE BITS datanamex AND datanamez
COMBINE BITS literal AND datanamez
```

Figure 127 Combine Fields Using AND

You can also express an AND where the result of the AND operation is placed in a third field. The contents of the original fields remain unchanged. These are the formats:

```
COMBINE BITS datanamew AND datanamex INTO datanamez
COMBINE BITS literal AND datanamex INTO datanamez
```

Figure 128 Express an AND

A 1- to 256-byte field of any data type can be combined using AND with a 1- to 256-byte field of any data type. The sending and receiving fields must be of equal length.

An AND operation works in this manner: a bit position in the receiving field is set to 1 if the corresponding bit positions in both the sending and receiving fields contain a 1; otherwise, the receiving bit is set to 0. You would probably use AND to a field when you want to turn bits off (set to 0).

Examples of AND operations:

```
COMBINE BITS DATA1 AND DATA2
```

Assume DATA1 is a 3-byte field containing the value (hex notation) of 010101. DATA2 is a 3-byte field containing the value of 000011. After the AND operation, DATA2 contains the value 000001.

```
COMBINE BITS X'010101' AND DATA2
```

The same result occurs as in the previous example, except that a hexadecimal literal value modifies the receiving field.

```
COMBINE BITS X'010101' AND DATA2 INTO DATA3
```

In this example, the data in DATA2 is moved to DATA3 before the AND is performed. DATA2 remains unaltered.

A variable-length AND operation is not valid.

COMBINE BITS Using EXOR

To combine fields using EXOR, use the following formats:

```
COMBINE BITS datanamex EXOR datanamez
COMBINE BITS literal EXOR datanamez
```

Figure 129 Combine Fields Using EXOR

You can express an EXOR where the field to be altered is moved to another field before the EXOR takes place:

```
COMBINE BITS datanamew EXOR datanamex INTO datanamez
COMBINE BITS literal EXOR datanamex INTO datanamez
```

Figure 130 Express an EXOR

Instead of EXOR, you can use the term EXCLUSIVE OR.

You can use an EXCLUSIVE OR between two 1- to 256-byte fields of any data type. The sending and receiving fields must be of equal length.

An EXCLUSIVE OR operation works in this manner: a bit position in the receiving field is set to 1 if the corresponding bit positions in the sending and receiving fields are different; otherwise, the receiving field bit is set to 0.

```
COMBINE BITS DATA1 EXOR DATA1
```

Assume DATA1 is a 3-byte field with the value (hex notation) FFFFFFFF. Because the field is in an EXCLUSIVE OR operation with itself, the result is 000000. This is a useful technique for clearing a field to binary zeros.

```
COMBINE BITS X'0F0F0F' EXOR DATA2
```

Assume DATA2 is a 3-byte field containing the value (hex notation) of FFFFFFFF. After the EXCLUSIVE OR operation, DATA2 contains the value F0F0F0.

```
COMBINE BITS X'0F0F0F' EXOR DATA2 INTO DATA3
```

In this example, the data in DATA2 is moved to DATA3 before the EXOR is performed. DATA2 remains unaltered.

```
COMBINE BITS DATA1 EXOR DATA2
COMBINE BITS DATA2 EXOR DATA1
COMBINE BITS DATA1 EXOR DATA2
```

This series of EXCLUSIVE OR operations causes a field or record to be exchanged with another field or record. After the end of the operations, the DATA1 field holds the original contents of the DATA2 field and the DATA2 field holds the original contents of the DATA1 field. This has been accomplished without the use of an intermediate work or hold area.

The fields involved in the EXCLUSIVE OR operation can be indexed:

```
COMBINE BITS DATA1 (INW) EXOR DATA2 (INX)
```

A variable length EXCLUSIVE OR operation is not valid.

MOVE CORRESPONDING Qualifier to Qualifier

This MOVE operation can be requested only if you are using qualified data names in your program and have specified OPTION QLF or OPTION QUALIFIERS. For more information, see [Chapter 5: Data Name Qualification](#). MOVE CORRESPONDING allows you to move a number of fields at the same time. The format is:

```
MOVE {CORRESPONDING|CORR} qualifier TO qualifier [SUPPRESS]
```

Figure 131 MOVE CORRESPONDING Qualifier to Qualifier

A qualifier is any file name, table name, array name, or named WORKAREA.

The sending and receiving fields in the move must have different qualifiers and at least one identical data name.

```
FILE SLSFILE FB 80 STATUS EOFFLAG
  STORENAME 25
  MANAGER 20
  CITY 15
  DAILYSLs 9 NU 2
  DATE 6 NU
TABLE SLSTBL F 40 ENTRY TBENT STATUS TBSTAT
  STORENAME 25
  DAILYSLs 9 NU 2
  DATE 4 NU
```

```
TABLOAD:
  READ SLSFILE
  .
  .
  MOVE CORRESPONDING SLSFILE TO SLSTBL
  STORE SLSTBL
  .
  .
```

In this example, all fields in SLSFILE that have a CORRESPONDING field in SLSTBL are moved. Three moves are accomplished simultaneously:

- SLSFILE.STORENAME is moved to SLSTBL.STORENAME.
- SLSFILE.DAILYSLs is moved to SLSTBL.DAILYSLs.
- SLSFILE.DATE is moved to SLSTBL.DATE.

The sending and receiving fields do not have to be of the same data type, but must conform to the standard rules for moves. For more information, see [Chapter 3: Syntax Rules](#).

Avoid defining overlapping fields when using MOVE CORRESPONDING. The order of the MOVEs can be unpredictable.

When you use MOVE CORRESPONDING statements in your program, the generated program listing shows the individual MOVE statements that were accomplished by the MOVE CORRESPONDING statement. If you do not want to see each MOVE statement, code:

```
MOVE CORRESPONDING SLSFILE TO SLSTBL SUPPRESS
```

or:

```
MOVE CORRESPONDING SUPPRESS SLSFILE TO SLSTBL
```

Only the MOVE CORRESPONDING statement is printed on the program listing.

Without SUPPRESS:

```
MOVE CORR SLSFILE TO SLSTBL
```

Program listing:

```
      MOVE CORR SLSFILE TO SLSTBL
INSERT  MOVE SLSFILE.STORENAME TO SLSTBL.STORENAME
INSERT  MOVE SLSFILE.DAILYSLs TO SLSTBL.DAILYSLs
INSERT  MOVE SLSFILE.DATE TO SLSTBL.DATE
```

With SUPPRESS:

```
MOVE CORR SLSFILE TO SLSTBL SUPPRESS
```

Program listing:

```
MOVE CORR SLSFILE TO SLSTBL SUPPRESS
```

MOVE UNSIGNED | ABSOLUTE | NORMALIZED

The MOVE statement allows for UNSIGNED, ABSOLUTE, and NORMALIZED fields. Datanamex and datanamey can be only binary, character, numeric, or packed decimal type fields.

UNSIGNED	A packed-like field in which the rightmost half-byte is a digit instead of a sign. For example, an unsigned field can contain X'123456'.
ABSOLUTE	A positive field. For numeric and packed decimal fields, the sign is a hex F.
NORMALIZED	A field whose sign is hex C if it is positive or hex D if it is negative.

The syntax for the MOVE statement is:

```
MOVE [UNSIGNED] datanamex TO [UNSIGNED|ABSOLUTE|NORMALIZED] datanamey
```

Figure 132 MOVE UNSIGNED|ABSOLUTE|NORMALIZED

The MOVE command adjusts for differing numbers of digits to the right of the decimal point in the sending and receiving fields.

The following examples show variations of the MOVE statement:

```
MOVE UNSIGNED datanamex TO datanamey
```

Figure 133 Variations of the MOVE Statement—Example 1

An unsigned packed number is converted to whatever type of number is contained in the receiving field.

```
MOVE datanamex TO ABSOLUTE datanamey
```

Figure 134 Variations of the MOVE Statement—Example 2

A number is forced to be positive in the receiving field.

```
MOVE UNSIGNED datanamex TO ABSOLUTE datanamey
```

Figure 135 Variations of the MOVE Statement—Example 3

This is similar to the previous example, but the sending field is an unsigned packed decimal number.

```
MOVE datanamex TO NORMALIZED datanamey
```

Figure 136 Variations of the MOVE Statement—Example 4

The sign of the receiving field is forced to be either C (for positive) or D (for negative).

```
MOVE UNSIGNED datanamex TO NORMALIZED datanamey
```

Figure 137 Variations of the MOVE Statement—Example 5

This example is similar to the last example, but the sending field is an unsigned packed decimal number.

```
MOVE datanamex TO UNSIGNED datanamey
```

Figure 138 Variations of the MOVE Statement—Example 6

This example converts a number into an unsigned packed decimal field.

The following table shows the valid field types for each type of MOVE command.

Dataname x	Field Type	Dataname y	Field Type
standard	BI, NU, PD	UNSIGNED	CH, PD
standard	BI, NU, PD	ABSOLUTE	BI, NU, PD
standard	BI, NU, PD	NORMALIZED	NU, PD
unsigned	CH, PD	standard	BI, NU, PD
unsigned	CH, PD	UNSIGNED	CH, PD
unsigned	CH, PD	ABSOLUTE	BI, NU, PD
unsigned	CH, PD	NORMALIZED	NU, PD

Chapter 11: Using Arithmetic Commands

VISION:Results supports addition, subtraction, multiplication, division, and exponentiation. You can code separate statements for each operator or you can combine operators in a single statement. Calculations can be done anywhere in your program, during regular or special (ON CHANGE, ON FINAL, ON END OF, and so on) logic.

The format of an arithmetic statement is as follows:

```
datanamez= {datanamew|literal} arithmetic operator {datanamex|literal}
```

Figure 139 Format of an Arithmetic Statement

Valid Arithmetic Operators

Neither the datanamew nor datanamex field is altered during the operation. The result of the operation is placed in datanamez.

The valid arithmetic operators and operations are:

- + ADD datanamex or literal to datanamew or literal. The result is stored in datanamez.
- SUBTRACT datanamex or literal from datanamew or literal. The result is stored in datanamez.
- * MULTIPLY datanamex or literal by datanamew or literal. The result is stored in datanamez.
- / DIVIDE datanamex or literal into datanamew or literal. The result is stored in datanamez.
- ** Raise datanamew or literal to a power datanamex or literal. The result is stored in datanamez.

Fields in the arithmetic operation do not have to be of the same size, data type, or have the same number of decimal positions. Arithmetic can be performed on numeric (NU), packed (PD), or binary (BI) data. The data conversion and decimal alignment are automatically handled.

```
AMOUNT3=AMOUNT2+AMOUNT1
AMOUNT3 = AMOUNT2 + AMOUNT1
BALANCE=AMOUNT*1.10
FOUR=2+2
AMOUNTX = AMOUNTY - 1,293.55
NEG=AMOUNT*-1
COUNTER=COUNTER+1
AVG=AMOUNT/COUNT
MTHBAL(INX)=MTHSAL(INX)*1.10
INT=SIZE**2/12
```

Indexing

Indexing can be used to allow calculations using entries in a table or segments in a record. For a full description of indexing, see [Chapter 14: Using Procedural Commands](#).

Example:

```
AMOUNT (INY) = AMOUNT2 (INY) + AMOUNT3 (INY)
MTHBAL (INX) = MTHSAL (INX) * 1.10
MTHSAL = DAILYSAL (INW) + .50
```

Rounding

The result of an arithmetic operation can be rounded:

```
AMOUNT1=AMOUNT2 * 1.10 ROUNDED
```

Assume AMOUNT2 is a 4-byte packed field with two decimals and contains the value 512.55, which is being multiplied by 1.10 (AMOUNT2 is being increased by 10%). The product of this multiplication is 563.8050. Without rounding, the result placed in AMOUNT1 (which, for this example, is defined as a 5-byte packed 2-decimal field), is 563.80.

If rounding is needed, the word ROUND or ROUNDED is coded with the arithmetic expression and the figure 563.81 would be placed in AMOUNT1.

When you specify that the result be rounded, the result cannot be defined as having over four decimal positions.

Exponentiation

VISION:Results supports exponentiation. A double asterisk (**) indicates that exponentiation is to be performed. The two fields involved are converted to floating point. Then, the exponentiation using the floating point values is performed, and it converts the result back to decimal. If the result of the calculation is too large to handle, an abend results, with the error indicating that the result is too large.

Positive and negative exponents are supported as well as integer and decimal exponents.

When raising a non-integer number to a power, the result might not be precise. This occurs because when converting from decimal to floating point, a small degree of accuracy is sometimes lost.

If you elected to not allow exponentiation when you installed, using exponentiation (**) results in an error message indicating that ** is not valid in your system.

Order of Computation

Formulas or calculations can be expressed in one statement. Any number of arithmetic operators and operands can be combined to produce a result:

```
AMOUNT1 = AMOUNT2 * 1.10 + 5.20
```

The value in AMOUNT2 is multiplied by 1.10, then 5.20 is added to the product that is finally placed in the AMOUNT1 field.

Computation is performed according to the rules of algebra. Each of the operators is assigned a priority.

Operation	Priority
Exponentiation (**)	1
Multiplication and division (*, /)	2
Addition and subtraction (+, -)	3

This priority determines the order in which operations are performed. Priority 1 is performed ahead of priority 2, and priority 2 ahead of priority 3.

Operations with the same priority are performed in the order in which they are coded (except for exponentiation). Sequential exponentiation operators are processed from right to left.

Parentheses can be used to change the order of computation.

Again, as in arithmetic operations, only the final result field of a calculation is changed. The fields taking part in the arithmetic do not change. All operations take place in work areas.

```
AMOUNT1=AMOUNT2 + AMOUNT3 * 1.10
```

The AMOUNT3 value is multiplied by 1.10, then this result is added to the AMOUNT2 value. The final result is placed in AMOUNT1.

```
CALC=AMT**5**2
```

5 is raised to the power 2 (squared) and then AMT is raised to the result of the previous calculation (AMT**25). The final result is placed in CALC.

```
AMOUNT1=(AMOUNT2+AMOUNT3)*1.10
```

Because there are parentheses around the AMOUNT2+AMOUNT3 operation, it is performed first. This result is then multiplied by 1.10 and the final result placed in AMOUNT1.

```
AMOUNT1=(AMOUNT2+AMOUNT3)*1.10   ROUNDED
```

This is the same as the previous example, except that before the result is placed in the AMOUNT1 field, it is rounded.

You can nest parentheses. The execution of each operation proceeds from left to right, innermost to outermost parentheses:

```
AMOUNT1=AMOUNT2 * 1.10 - ((AMOUNT3 + 10)/3)
```

The sequence of execution here is as follows:

1. The AMOUNT2 value is multiplied by 1.10.
2. 10 is added to the AMOUNT3 value.
3. The result of the addition is divided by 3.
4. The result of the division is subtracted from the result of the multiplication.
5. The final result is placed in AMOUNT1.

Arithmetic Restrictions

- All arithmetic operations are performed using internal work areas capable of handling a maximum of 30 digits (21 whole numbers and 9 decimal digits), in packed decimal format.
- When the number of digits involved in the calculation exceeds the maximum allowable limit, all excess digits are truncated. For example, if the result of an addition (+), subtraction (−), multiplication (*), or division (/) operation requires more than 21 whole numbers or 9 decimal digits, truncation occurs. If the operation is exponentiation (**), an execution time abend (USER 299) occurs. In addition, an error message appears showing the contents of the fields involved.
- A negative number can be raised to a power. However, if a fractional exponent is specified, an execution time abend results, indicating that this is invalid.

Chapter 12: Using the SORT Command

The SORT statement allows you to sort input file records or work area records in any sequence before they are reported on, printed, or written to an output file.

The records being sorted can be from sequential (fixed, variable, spanned) files, VSAM files, [ISAM](#) files, database files, or records built in a work area in your program.

You can code your SORT statement at the very beginning of your procedure logic to sort all input records, or you can select certain records or build work area records before sorting.

The SORT statement can be logically divided into two phases:

- A release phase that precedes the SORT statement
- A return phase that follows the SORT statement

```
CONTROL ACCOUNT
IF ACCOUNT EQ 'AO' THRU 'FO' NEXT ELSE REJECT ENDIF
  SORT ARFILE USING ACCOUNT
  LIST ACCOUNT NAME BALANCE
ON CHANGE IN ACCOUNT
  LIST SUM BALANCE WITH 2 BEFORE AND 2 AFTER
ON FINAL
  LIST SUM BALANCE WITH 2 BEFORE
```

Figure 140 SORT Statement

All procedure logic statements above the SORT statement are executed repetitively, releasing records to the SORT statement one at a time as they are selected. When VISION:Results recognizes the end of the file on the file being sorted, it returns records from the SORT, one at a time, and invokes the procedure logic statements following the SORT statement, with the exception of any ON CHANGE IN, ON FINAL, or ON END OF SORTING and their subordinate statements. VISION:Results continues to return records from the SORT until the sorted records have been exhausted or the user stops the SORT. It then terminates program execution.

There are three formats of the SORT statement:

- Sorting a file
- Sorting a file ...until
- Sorting a dataname ...until

VISION:Results uses standard SORT field formats that allow you to instruct SORT to use portions of fields containing dates as a sort key. To use these Y2K SORT field types, you must set up VISION:Results to REDEFINE the date fields and provide definitions for the field subsets you want to use. Then, use the redefined names in the SORT statement syntax followed by the new SORT Y2K field format reserved word(s) PD0, Y2C, Y2Z, Y2P, or Y2D.

In addition, the SORT command can be dynamically invoked using the RELSORT and RETSORT commands. For more information, see [RELSORT and RETSORT Commands on page 263](#).

Sorting a File

You can sort an input file by one or more fields (keys). The sorted records can be in ascending or descending order. The SORT filename in your VISION:Results program causes your procedure logic prior to the SORT statement to be executed for each input record until the end of the file is sensed. Records not rejected are passed to SORT and arranged in order. The SORT statement then passes one sorted record at a time to the remaining procedure logic.

Format 1 Syntax

```
SORT filename [nK]
USING ! dataname {PD0 | Y2C | Y2Z | Y2D | Y2P} {A | D}! [WORK nnn]
```

Figure 141 Format for Sorting a File

Format 1 Example

```
SORT ARFILE 36K USING ACCOUNT TRANS
SORT ARFILE USING ACCOUNT TRANS NAME
```

File Name

When using Format 1, the SORT command is immediately followed by the file name. The file name specifies where to find the records to be sorted.

nK

Following the file name, you can optionally supply the amount of memory (in thousands) to be given to sort. You specify this in the form nK, where n is an integer from 12 to 999. If not specified, the system does one of the following:

- z/OS Defaults to a minimum of 36K to a maximum 100K. These values can be overridden by this DYLINK parameter SORTMEM=(minimum, maximum). For details, see the *Advantage VISION:Results for z/OS Installation Guide*.
- VSE Defaults to the available unused partition space.

USING dataname

Following the file name (and the nK, if supplied) is the keyword USING, which always precedes the data names/field names (sort keys). The sort keys describe the fields to be sorted and the requested sequence of the sort. Code the data names from left to right in decreasing order of sequence (from the most major to the most minor). The number of data names (sort keys) following the USING keyword is approximately 15.

`SORT ARFILE USING ACCOUNT TRANS`

ACCOUNT is the major key and TRANS is the minor key. All transaction numbers are sorted within each sorted account code.

ACCOUNT	TRANS
CO	1234567
CO	1245678
CO	1870000
DE	0010000
FG	0000001
FG	0000003
FG	0008976

USING dataname for Y2K Formats

{PD0|SPT|Y2C|Y2Z|Y2P|YR2|SYR|Y2D|YR1|CMp|PMp|FMp}

The SORT-defined Y2K field formats are:

Code	Format	Description
PD0 or SPT for CA-SORT 7.1	Partial packed decimal	Looks at parts of PD fields. The sign (last) and first digits of the field are ignored. Only the underscored text would be used by SORT for the following: x'0 <u>1</u> 02' or '0 <u>9</u> 9C'. This can be used to individually pass the month, day, and year portion of a PD date field.
Y2C, Y2Z , YR2 for CA-SORT 7.1 , or Y2S for CA-SORT 8.2 for VSE	2-byte, 2-digit year for a CH or NU field 2-byte, 2-digit year for a CH, NU, or BI field 2-byte, 2-digit year for a CH, NU, or BI field	Only the digit portion of each of the two bytes is used; the zone is ignored. Only the underscored text would be used by SORT as the year value for the following: x'F <u>2</u> F <u>3</u> '.
Y2P or SYR for CA-SORT 7.1	2-byte, 2-digit year for a PD field	The sign (last) and first digits of the field are ignored. Only the underscored text would be used by SORT as the year value for the following: x'0 <u>9</u> 9C'.
Y2D, YR1 for CA-SORT 7.1, or YR1 for CA-SORT 8.2 for VSE	1-byte, 2-digit year for a PD field 1-byte, 2-digit year for a BI field	The digits would be passed as an unsigned PD or BI field. SORT would use the following underscored text as the year value: x' <u>9</u> 9'.

Code	Format	Description
CMp , PMp , FMp for CA-SORT 7.1	date fields in CH, PD, BI, respectively	p is 0-9, A or B. The values of Mp represent dates in the following formats: M0 = dddyy M1 = yyddd M2 = mddyy M3 = yymmdd M4 = yyddmm M5 = ddmmyy M6 = dddyyy M7 = yyyddd M8 = mddyyyy M9 = ddmyyyy MA = yyyyymmdd MB = yyyyddmm

- **PD0** (partial packed decimal) or **SPT** for CA-SORT 7.1 to look at parts of PD fields.

The sign (last) and first digits of the field are ignored. Only the underscored would be used by SORT for the following: x'0102' or '099C'. This can be used to individually pass the month, day and year portion of a PD date field.

- **Y2C**, **Y2Z** (2-byte, 2-digit year for a CH or NU field), **YR2** for CA-SORT 7.1 (2-byte, 2-digit year for a CH, NU, or BI field), or **Y2S** for CA-SORT 8.2 for VSE (2-byte, 2-digit year for a CH, NU, or BI field).

Only the digit portion of each of the two bytes is used; the zone is ignored. Only the underscored would be used by SORT as the year value for the following: x'F2F3'.

- **Y2P** or **SYR** for CA-SORT 7.1 (2-byte, 2-digit year for a PD field).

The sign (last) and first digits of the field are ignored. Only the underscored would be used by SORT as the year value for the following: x'099C'.

- **Y2D**, **YR1** for CA-SORT 7.1 (1-byte, 2-digit year for a PD field), or **YR1** for CA-SORT 8.2 for VSE (1-byte, 2-digit year for a BI field).

The digits would be passed as an unsigned PD or BI field. SORT would use the following underscored text as the year value: x'99'.

- **Cmp, PMp, Fmp** for CA-SORT 7.1 (date fields in CH, PD, BI, respectively), where **p** is 0-9, A or B. The values of **Mp** represent dates in the following formats:

M0 = dddyy M3 = yymmdd

M1 = yyddd M4 = yyddmm

M2 = mmddyy M5 = ddmmyy

```
FILE SAMPFL
  DATEPD PD 4 VALUE 102099 ; X'0102099C'
  DATEMO CH 2 1           ; X'0102'
  DATEDA .CH 2 2           ; X'0209'
  DATEYY CH 2 3           ; X'099C'
  .
  .
  .
SORT SAMPFL USING DATEMO PD0 DATEDA PD0 DATEYY Y2P
```

A or D

After each data name, you can optionally supply a code letter to specify whether the records are to be sorted in ascending (A) or descending (D) order for that field. The system default is ascending (A) and does not need to be specified.

```
SORT ARFILE USING ACCOUNT A TRANS D
SORT ARFILE USING ACCOUNT TRANS D
SORT ARFILE USING ACCOUNT D TRANS
```

WORK n

Note: This section applies to VSE only.

If you are operating under VSE and you want to use more than one sort work file (the default is one sort work file), you can specify how many you require using the keyword **WORK** in the **SORT** statement.

```
SORT FILE USING ACCOUNT TRANS WORK 3
```

The number specified, **WORK n**, is directly associated with the number of sort work files, **SORTWK1-n**, and **DLBL/EXTENT JCL** statements required. **WORK 1** or the default requires **DLBL/EXTENT** for **SORTWK1**. **WORK 3** requires **JCL** for **SORTWK1**, **SORTWK2**, and **SORTWK3**. For more information, see [VSE JCL Requirements on page 272](#).

A sample **VISION:Results** program containing the **SORT** statement is shown below. In this example, the entire file is sorted in account code sequence.

Example 1

```

FILE ARFILE FB 352 5280
ACCOUNT 2 182 NAME 25 85 TRANS 7 4
BALANCE 5 170 PD 2 A
CONTROL ACCOUNT
  SORT ARFILE USING ACCOUNT <----
  LIST SUPPRESS ACCOUNT TRANS NAME BALANCE
ON CHANGE IN ACCOUNT
  LIST SUM BALANCE WITH 2 BEFORE AND 2 AFTER
T1 'VISION:Results SORT REPORT 1' WITH 2 AFTER

```

Figure 142 SORT Statement [Example 1](#)

The following sequence takes place:

1. The automatic cycle of VISION:Results reads a record from ARFILE.
2. VISION:Results checks for end of file condition. If end of file, go to Step 5.
3. The record is released to the sort using the SORT statement.
4. VISION:Results automatically returns to Step 1.
5. A record is returned from the sort using the SORT statement.
6. VISION:Results automatically checks to see if the sorted records are exhausted; if exhausted, go to Step 9.
7. VISION:Results automatically checks to see if a control break has occurred. If so, the LIST statement subordinate to the ON CHANGE IN is executed.
8. A report line is printed using the LIST statement; go to Step 5.
9. End of job; any ON CHANGE logic is processed and the program is terminated.

This VISION:Results program, besides giving additional information on the report, also selects specific records to be sorted (that is, records where ACCOUNT is equal to NA or EO through IO, inclusive).

Example 2

```

FILE ARFILE FB 352 5280
ACCOUNT 2 182 NAME 25 85 TRANS 7 4
BALANCE 5 170 PD 2 A
WORKAREA
  COMMENT 3 (BALANCE'GE $300.00'OR'LT $0.00) VALUE ' ' REINIT
CONTROL ACCOUNT
  IF ACCOUNT EQ 'NA' 'EO' THRU 'IO' NEXT ELSE REJECT ENDIF
  SORT ARFILE USING ACCOUNT TRANS <----
  IF BALANCE GE 300
    MOVE '***' TO COMMENT GOTO LISTDET ENDIF
  IF BALANCE LT 0
    MOVE '---' TO COMMENT ENDIF
LISTDET:
  LIST SUPPRESS ACCOUNT TRANS NAME BALANCE COMMENT
ON CHANGE IN ACCOUNT
  LIST SUM BALANCE WITH 2 BEFORE AND 2 AFTER
ON FINAL
  LIST SUM BALANCE WITH 2 AFTER
  LIST 'NUMBER OF RECORDS PRINTED:' AT ACCOUNT
  TALLY AT BALANCE
T1 'VISION:Results SORT REPORT 2' WITH 2 AFTER

```

Figure 143 SORT Statement [Example 2](#)

The following sequence takes place:

1. The automatic cycle of VISION:Results reads a record from ARFILE. If end of file, go to Step 5.
2. The record is rejected if it is not an NA or EO through IO account code; go to Step 1.
3. The record is released to the sort using the SORT statement.
4. VISION:Results automatically returns to Step 1.
5. A record is returned from the sort using the SORT statement.
6. VISION:Results checks to see if the sorted records are exhausted; if exhausted, go to Step 11.
7. If the BALANCE field is greater than or equal to 300, COMMENT field is set to `***'; go to Step 9.
8. If the BALANCE field is less than zero, COMMENT field is set to `---.'
9. VISION:Results automatically checks to see if a control break has occurred. If so, the LIST statement subordinate to the ON CHANGE IN is executed.
10. A report line is printed using the LIST statement; go to Step 5.
11. End of job; ON CHANGE and ON FINAL logic is processed and the program is terminated.

The rest of this chapter deals with the more advanced techniques of the SORT statement—[ON END OF SORTING](#), [SORT A FILE ... UNTIL](#), and [Sorting a Data Name ... UNTIL](#).

ON END OF SORTING

When the sorted input is exhausted, processing of the program is terminated. Any ON CHANGE IN logic and ON FINAL logic is executed prior to closing the files. You can intervene to do some additional wrap-up logic. This is accomplished using the ON END OF SORTING command. Once used, you have complete control. You must terminate processing using the STOP command.

ON END OF SORTING

Figure 144 ON END OF SORTING Command

[Example 3](#) demonstrates the use of the ON END OF SORTING command to write a trailer record when the records returned from the sort have been exhausted.

Example 3

```

FILE ARFILE FB 352 5280
  ARREC 352   NAME 25 85
  BALANCE 5 170 PD 2 A   ACCOUNT 2 182
FILE OARFILE FB 352 5280 OUTPUT FROM ARFILE
WORKAREA
  TBALANCE 5 PD 2 VALUE 0
CONTROL ACCOUNT
SORT ARFILE USING ACCOUNT <----
IF ACCOUNT EQ 'AO' THRU 'PZ'
  TBALANCE=BALANCE+TBALANCE ENDIF
WRITE OARFILE
LIST SUPPRESS ACCOUNT BALANCE NAME

ON END OF SORTING <----
  MOVE SPACES TO ARREC
  MOVE 'ZZ' TO ACCOUNT
  MOVE TBALANCE TO BALANCE
  WRITE OARFILE
STOP
ON CHANGE IN ACCOUNT
  LIST SUM BALANCE WITH 2 BEFORE AND 2 AFTER
T1 'REPORT 3' WITH 2 AFTER

```

Figure 145 SORT Statement [Example 3](#)**Example 4**

[Example 4](#) demonstrates the use of the ON END OF SORTING command where both input files must be exhausted before the program terminates. A step-by-step description follows.

```
FILE FILEA FB 352 5280 STATUS FILEASTAT
  READKEY 7 4
.
.
FILE FILEB FB 352 10560
  SORTKEY 7 8
.
.
WORKAREA SORTSTAT 1
.
.
READ FILEB
SORTREC:
  SORT FILEB USING SORTKEY      <----
  ON ONE
    READ FILEA
  ENDONE
CHECK:
  IF SORTKEY EQ READKEY
    GOTO MATCHIT ENDIF
  IF SORTKEY LT READKEY
    AND SORTSTAT NE 'E'
    GOTO SORTREC ENDIF
  ; ASSUMED GT ...
READ FILEA
  IF FILEASTAT EQ 'E' AND SORTSTAT EQ 'E'
    STOP ENDIF
GOTO CHECK
MATCHIT:
.
.
ON END OF SORTING              <----
  IF FILEASTAT EQ 'E' STOP ENDIF
  MOVE 'E' TO SORTSTAT
  MOVE X'FFFFFFFFFFFFFF' TO SORTKEY
  GOTO CHECK
ON CHANGE IN ....
.
.
```

Figure 146 SORT Statement [Example 4](#)

The following sequence takes place:

1. A record is read from FILEB.
2. VISION:Results checks for an end of file condition; if end of file, go to Step 5.
3. The record is released to the sort using the SORT statement.
4. VISION:Results returns to Step 1.
5. A record is returned from the sort using the SORT statement.
6. One-time-only read of FILEA; if you have been here before, go directly to Step 7.
7. The sorted record's SORTKEY is compared to FILEA's READKEY; if they are equal, go to Step 12.
8. The sorted record's SORTKEY is compared to FILEA's READKEY; if SORTKEY is less and the sorted records have not been exhausted, go to Step 5.

9. Read a record from FILEA.
10. Check FILEA's status indicator FILESTAT and the sorted record's status indicator SORTSTAT to see if end of file has been reached on both. If end of file has been reached on both files, STOP; otherwise, go to Step 7.
11. Match logic takes place here. VISION:Results returns to Step 5 when it encounters the ON END OF SORTING statement.
12. Entered only (ON END OF SORTING) when sorted records are exhausted. FILEA's status is checked for end of file. If end of file has been reached, STOP.
13. The sorted record's status indicator is set to E to indicate an end of file condition.
14. High-values are moved to SORTKEY, so comparisons against the READKEY are always greater and FILEA continues to be processed until end of file is reached. If end of file has been reached, go to Step 7.

The same logic and constraints of the ON END OF SORTING apply to the sections that follow.

For further information about ON END OF SORTING considerations, see [Limitations and Constraints on page 268](#).

SORT A FILE ... UNTIL

You can terminate the input phase of the sort prior to reaching end of file on the file being sorted. Three approaches are possible. You can terminate the input phase using a data name, using the file name of the file being sorted, or using a file name other than the file being sorted.

Format 2 Syntax

```
SORT filename [nK]
USING ! dataname {PD0 | Y2C | Y2Z | Y2D | Y2P} {A | D}! [WORK nnn]

UNTIL {filename} [WORK n]
      {dataname}
```

Figure 147 Format for SORT a File... UNTIL

With the exception of the UNTIL {filename} {dataname}, all other keywords are described in [Sorting a File on page 238](#) and are not described here.

UNTIL

UNTIL is required when using this format of the SORT. It must follow all data names (sort keys) specified after the USING keyword in the SORT statement.

```
SORT ARFILE USING ACCOUNT TRANS UNTIL ...
```

Immediately following UNTIL, you can specify a file name or data name.

UNTIL filename

You can specify the file name of the file being sorted or the file name of another file. In either case, the input phase of sort terminates only when end of file is detected on the specified file.

The following two programs produce the same results.

Example 5

```
FILE ARFILE FB 352 5280
  ACCOUNT 2 182  BALANCE 5 170 PD 2 A
SORT ARFILE USING ACCOUNT          <----
LIST ACCOUNT BALANCE
T1 'REPORT 1' WITH 2 AFTER
```

Figure 148 SORT Statement [Example 5a](#)

```
FILE ARFILE FB 352 5280
  ACCOUNT 2 182  BALANCE 5 170 PD 2 A
SORT ARFILE USING ACCOUNT UNTIL ARFILE  <----

LIST ACCOUNT BALANCE
T1 'REPORT 1' WITH 2 AFTER
```

Figure 149 SORT Statement [Example 5b](#)

The first program uses Format 1 and the second program uses Format 2. The SORT statement in the second program has the UNTIL ARFILE, but this is not required. In both programs, VISION:Results terminates the input phase of the sort when end of file is detected on ARFILE. See [Figure 150](#). The flowchart logic applies to both of these program examples.

```
SORT ARFILE USING ACCOUNT
SORT ARFILE USING ACCOUNT UNTIL ARFILE
```

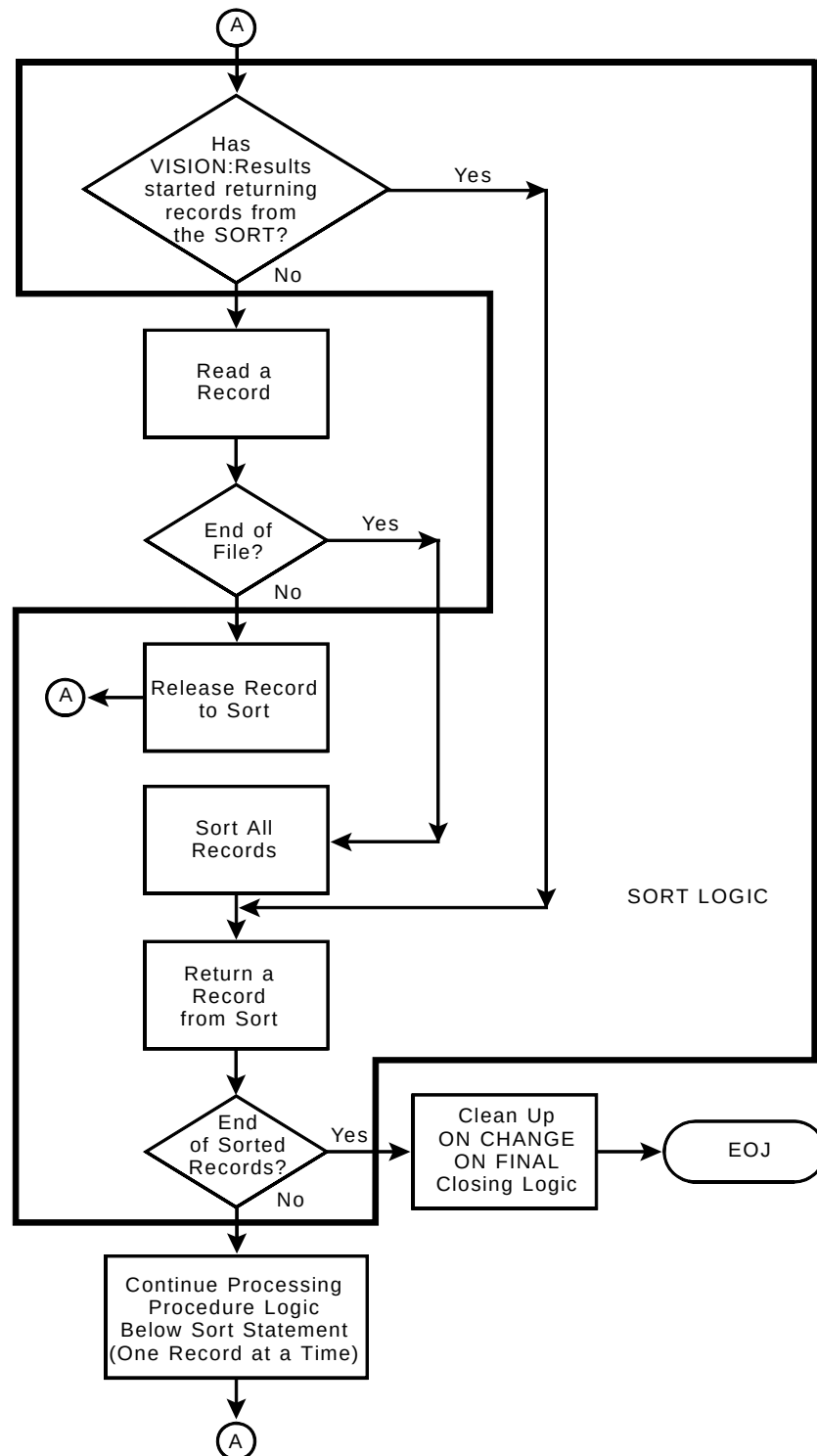



Figure 150 SORT Flow Diagram

Terminating the input phase using an end of file condition can be triggered from a second file.

[Example 6](#) demonstrates the merging of two similar files (block size is the only difference) and terminating the input phase of the sort when end of file is detected on the second file.

Example 6

```
FILE ARQ1FIL FB 352 5280 STATUS ARQ1EOF
  ARQ1REC 352  ACCOUNT 2 182  BALANCE 5 170 PD 2 A
FILE ARQ2FIL FB 352 10560
  ARQ2REC 352

READ ARQ1FIL
IF ARQ1EOF EQ 'E' GOTO READQ2 ENDIF
GOTO SORTREC
READQ2:
  READ ARQ2FIL
  MOVE ARQ2REC TO ARQ1REC
SORTREC:
  SORT ARQ1FIL USING ACCOUNT UNTIL ARQ2FIL      <----
  LIST ACCOUNT BALANCE
T1 'REPORT MERGE' WITH 2 AFTER
```

Figure 151 SORT Statement [Example 6](#)

[Figure 152, Merging Two Files With SORT Flow Diagram](#), illustrates the logic, both user and internal, in the above program.

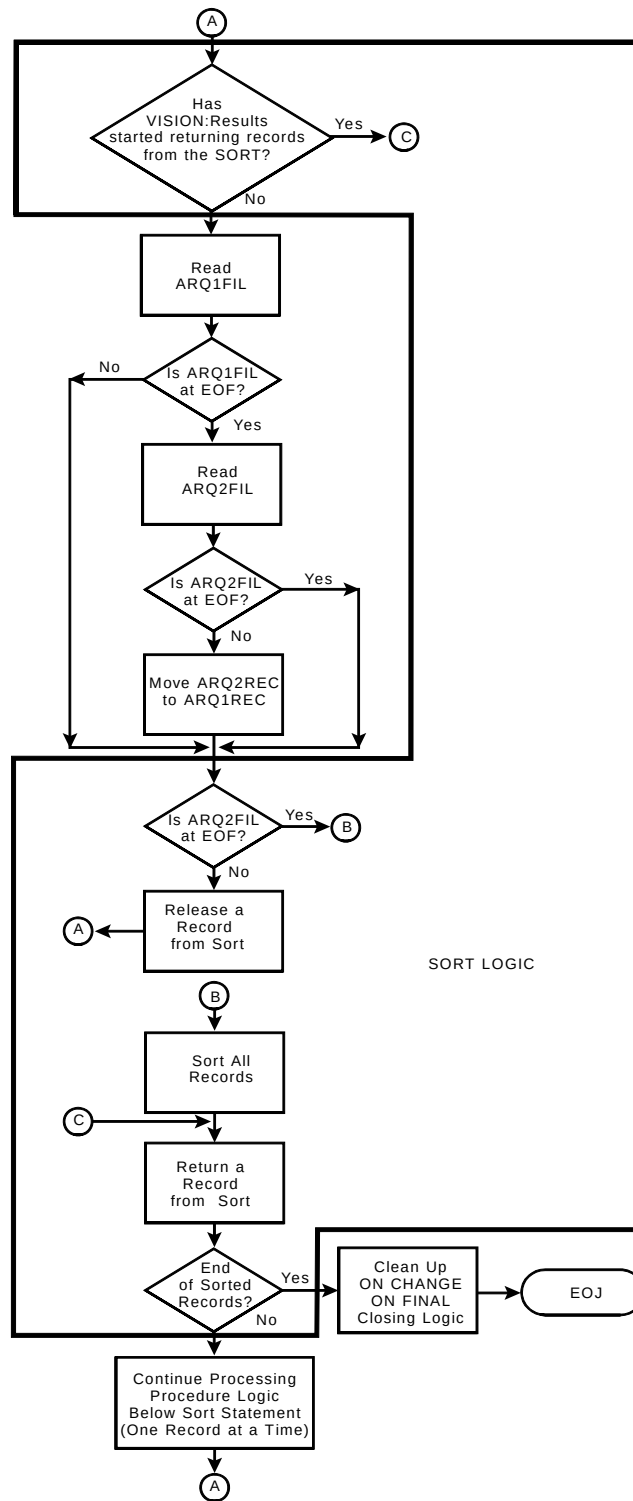


Figure 152 Merging Two Files With SORT Flow Diagram

UNTIL dataname

This section discusses the use of a data name after the UNTIL keyword in a SORT statement. The data name can be a field defined in a record or work area. You must manually set (MOVE) the data name's value to E and execute the SORT statement for the input phase of the sort to be terminated. You can also prematurely end the output phase of the sort by placing a value of C into the data name during the output phase of SORT. See [Example 10](#).

Example 7

Note: This example and [Figure 154](#) demonstrate the use of a data name (placed after the UNTIL keyword in the SORT statement) to end the input phase. When the account code is WO, the release phase of SORT terminates. The record with an account code of WO is not released to SORT (see [Figure 154](#), [SORT UNTIL Flow Diagram on page 253](#)). The logic of the program falls into the SORT statement after the E is moved to EOFEQE.

```
FILE ARFILE FB 352 5280
  ACCOUNT 2 182
  BALANCE 5 170 PD 2 A

WORKAREA
  EOFEQE 1

READ ARFILE
  IF ACCOUNT EQ 'WO'
    MOVE 'E' TO EOFEQE ENDIF

SORT ARFILE USING ACCOUNT <----
  UNTIL EOFEQE

LIST ACCOUNT BALANCE
TI 'REPORT 1' WITH 2 AFTER
```

Figure 153 SORT Statement

Note: The LIST logic (return phase) is not executed until the UNTIL EOFEQE is satisfied (equal to an E).

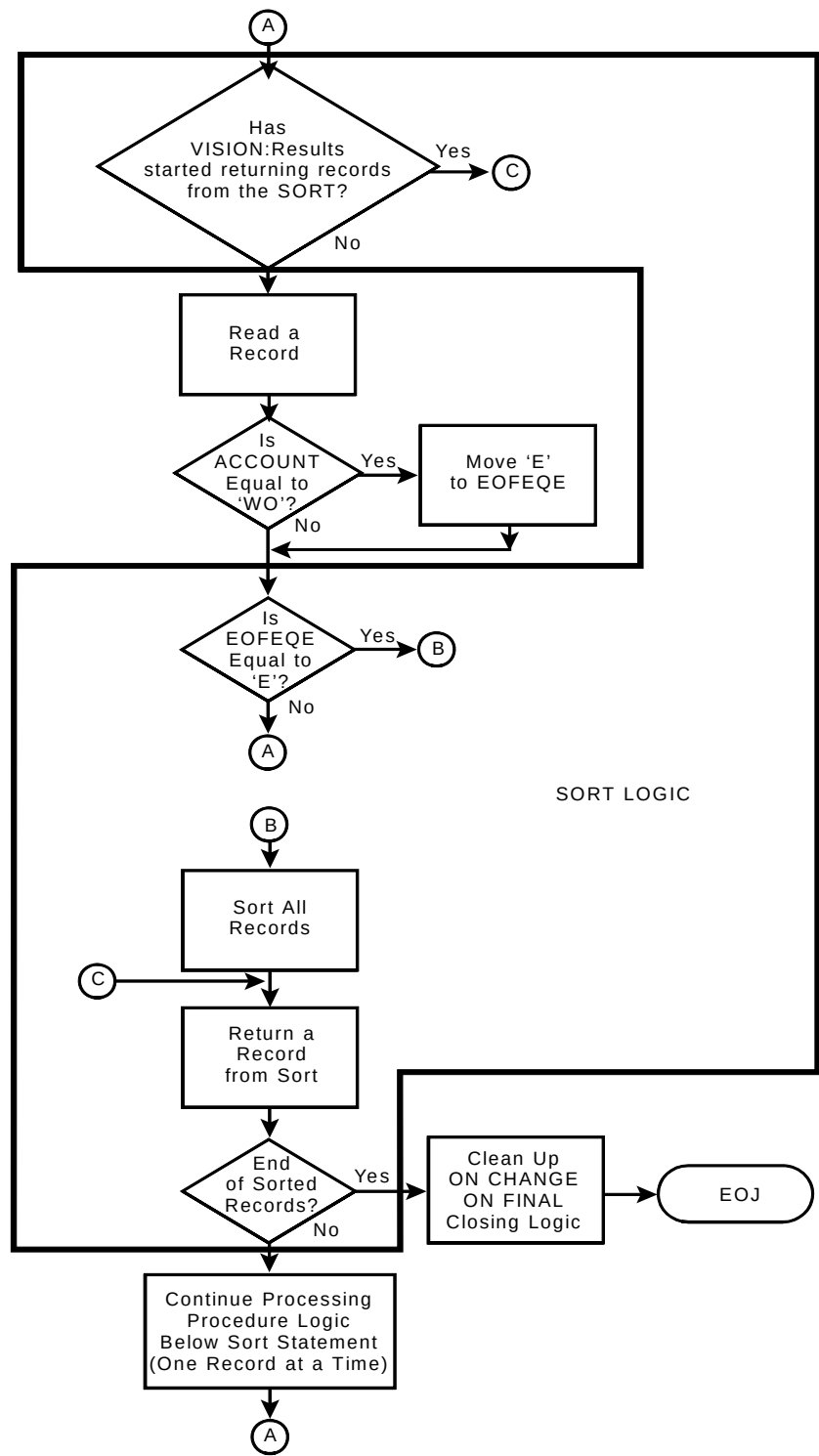


Figure 154 SORT UNTIL Flow Diagram

Once a data name (field name) is specified in the UNTIL of a SORT statement, you have control. VISION:Results cannot terminate the input phase of the sort itself. If the condition is not met and an E is not moved to the data name specified after the UNTIL, your program goes into an endless loop. z/OS users eventually time out. VSE users must have a computer operator cancel their job.

The only exception is if the data name used in the UNTIL of a SORT statement is associated with the STATUS keyword (shown in [Example 8](#)) of a file. If that file goes to end of file, it automatically is set to E by VISION:Results, causing the release phase of the SORT to terminate when the SORT command (or RELSORT command) is next executed.

Example 8

```
FILE ARFILE FB 352 5280 STATUS EOFEQE
  ACCOUNT 2 182
  BALANCE 5 170 PD 2 A

READ ARFILE
IF ACCOUNT EQ 'WO'
  MOVE 'E' TO EOFEQE ENDIF

SORT ARFILE USING ACCOUNT UNTIL EOFEQE  <----

LIST ACCOUNT BALANCE
TI 'REPORT 1' WITH 2 AFTER
```

Figure 155 SORT Statement [Example 8](#)

In the above program the release phase of the sort would terminate normally on one of two conditions:

- If a WO account code is detected.
- If an end of file is detected (no WO accounts are encountered on the file).

An E in a data name does not terminate the release phase of the SORT in and of itself. This is true even if the data name is associated with the STATUS keyword. The logic of the program must proceed to the SORT statement, as it does in [Example 9](#), for VISION:Results to terminate the release phase of SORT.

Example 9

```

FILE ARFILE FB 352 5280 STATUS EOFEQE
      BALANCE      5 170 PD 2 A
      ACCTCODE      2 182

READ ARFILE
IF ACCTCODE EQ 'WO'
    MOVE 'E' TO EOFEQE ENDIF
IF ACCTCODE NE 'XX'
    REJECT ENDIF

SORT ARFILE USING ACCTCODE UNTIL EOFEQE

LIST ACCTCODE BALANCE
TI 'REPORT 2' WITH 2 AFTER

```

Figure 156 SORT Statement [Example 9](#)

In [Example 9](#), an E is moved to the STATUS data name under the same conditions as in [Example 8](#). However, if there are no ACCTCODEs with a value of XX, the program goes into an infinite loop because the logic of the program never proceeds to the SORT statement.

The program shown in [Example 10](#) and [Figure 158, SORT UNTIL Dataname Flow Diagram on page 256](#) demonstrates the use of a data name (placed after the keyword UNTIL in the SORT statement) to prematurely end the output phase of SORT. When an account code of WO is encountered, the return phase of SORT terminates. SORT does not actually receive the value placed into EOFEQE until one additional SORT cycle has taken place. Any record with an account code of WO or greater is not listed on the report.

Example 10

```

FILE ARFILE FB 352 5280 STATUS AREOF
      ACCOUNT 2 182
      BALANCE 5 170 PD 2 A

WORKAREA
  EOFEQE 1

READ ARFILE
IF AREOF EQ 'E'
    MOVE 'E' TO EOFEQE ENDIF

SORT ARFILE USING ACCOUNT
      UNTIL EOFEQE

IF EOFEQE EQ 'C'
    STOP ENDIF

IF ACCOUNT EQ 'WO'
    MOVE 'C' TO EOFEQE
REJECT ENDIF
LIST ACCOUNT BALANCE

```

Figure 157 SORT Statement [Example 10](#)

The entire input file is sorted until end of file. Then the records are listed, as they are being returned from SORT, until a record with an account code of WO is detected. SORT then terminates immediately without listing the record.

One additional cycle must take place in order for SORT to receive the C placed into EOFEQE. This is required for the correct premature termination of SORT.

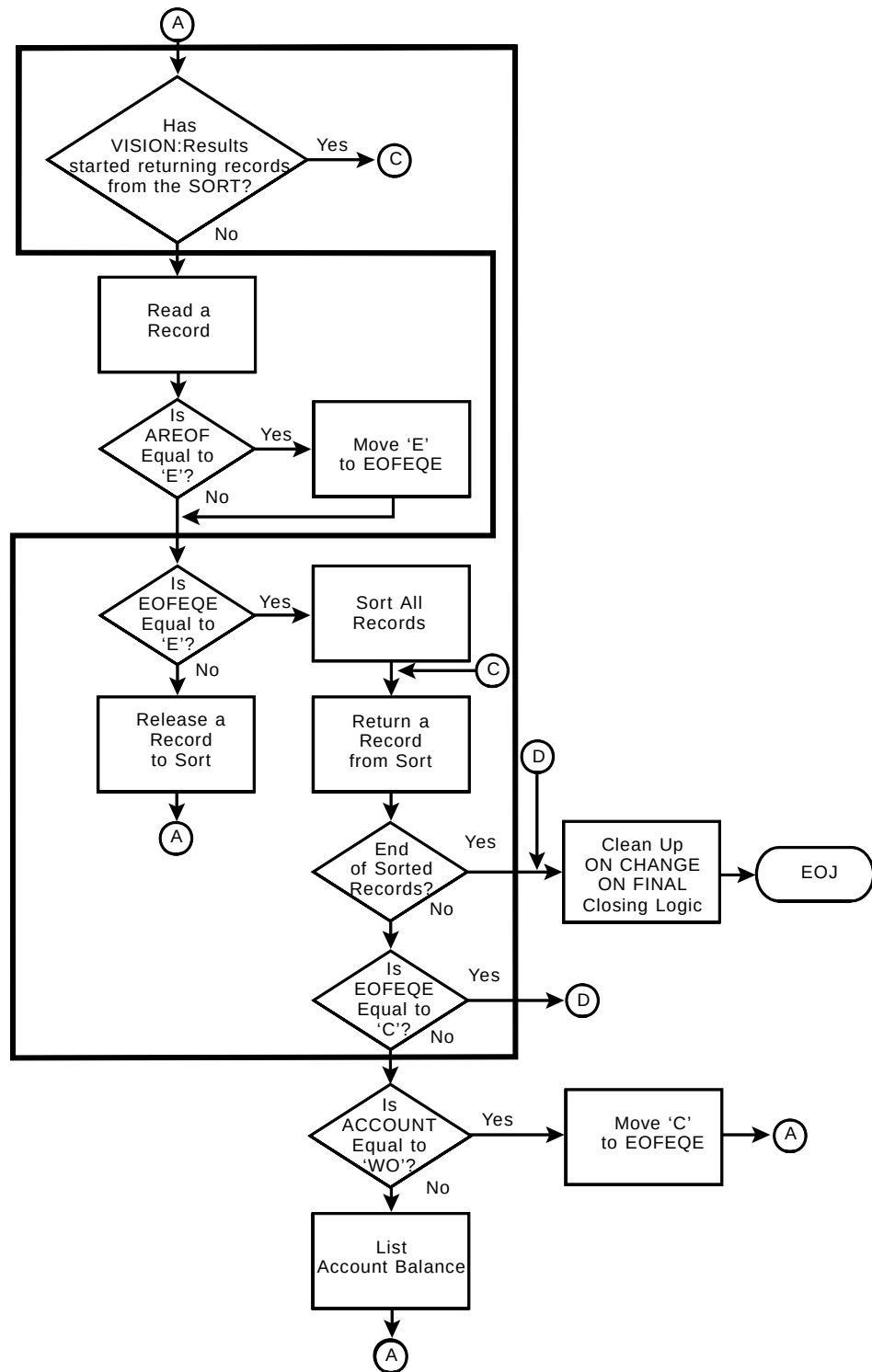


Figure 158 SORT UNTIL Dataname Flow Diagram

Sorting a Data Name ... UNTIL

The two previous sections described sorting entire input records in a file until the end of file is encountered or until certain qualifying conditions force the termination of the input phase of the sort. There are times, however, when the amount of information needed for processing or reporting does not require an entire input record or the information needed comes from any number of different files. VISION:Results provides this feature using the format shown below.

Format 3 Syntax

```
SORT dataname [recordsize] [nK]
USING ! dataname {PD0 | Y2C | Y2Z | Y2D | Y2P} {A | D}! [WORK nnn]

UNTIL {filename} [WORK n]
      {dataname}
```

Figure 159 Format for Sorting a Data Name ... UNTIL

Format 3 Example

```
SORT ACCOUNT USING ACCOUNT UNTIL ARFILE
SORT TRANS 200 USING ACCOUNT UNTIL ARFILE
SORT WACCT 50 USING WACCT UNTIL WEOF
```

When using the above format, the SORT command is immediately followed by a data name (field name). The data name can be a field in an input record or a field defined in a work area in your VISION:Results program. The data name tells VISION:Results where to find the beginning of the record area to be passed to the sort.

Record Size

In the previous examples, the length of the record to be sorted was always determined by the file's attributes and its record size. Because a record area (not a file) is being sorted, the length of the area to be sorted is determined by *one* of the following factors:

- The record size in the SORT statement that immediately follows the data name.

```
SORT WACCT 30 USING WACCT UNTIL ...
```

- The length of the data name, as defined in the VISION:Results program, if the record size is not specified.

```
SORT WACCT USING WACCT UNTIL ...
```

The above two SORT statements are not the same unless WACCT is defined with a length of 30.

The data name specified, after the SORT command, is the beginning of the record area to be passed to the sort but does not necessarily have to be a sort key.

For example, assume that the following were defined in a work area:

```
WORKAREA
WACCT      2
WBALANCE   5   PD   2   A
WTRANS     7
WDATE      6           D
WDESC      10
```

The following SORT statement starts with a beginning location of WACCT and releases a 30-byte record to the sort, sorting in WACCT sequence.

```
SORT WACCT 30 USING WACCT UNTIL ...
```

While the SORT statement starts with a beginning location of WACCT, it releases only a 2-byte record to the sort, the length of WACCT.

```
SORT WACCT USING WACCT UNTIL ...
```

USING

The keyword USING must always precede the data names (field names) of the fields (sort keys) on which you are sorting. VISION:Results validates the data names following the USING keyword to make sure that they are completely contained within the area to be sorted.

Taking the same work area plus an additional definition:

```
WORKAREA
WACCT      2
WBALANCE   5   PD   2   A
WTRANS     7
WDATE      6           D
WDESC      10
WCODE      3
```

Code the data names from the most major sort key to the most minor sort key.

```
SORT WACCT 33 USING WACCT WCODE UNTIL ...
```

UNTIL

The keyword UNTIL follows the sort keys and is required when using Format 2 or 3. Immediately following the UNTIL, you can specify either a file name or a data name. The concept behind the use of a file name or data name after the UNTIL is described in [SORT A FILE ... UNTIL on page 247](#). In addition, following are program examples and flowcharts for the above format.

UNTIL filename

[Example 11](#) illustrates the sorting of a partial record from an input file. The input phase of the sort is terminated when end of file is detected as specified in the SORT statement. A 100-byte record is released to the sort. In this example, the sort key (ACCOUNT) is not the first byte (position) of the record area to be released to the sort.

Example 11

```
FILE ARFILE FB 352 5280
  NAME 25 85  BALANCE 5 170 PD 2 A
  ACCOUNT 2 182

IF BALANCE GE 300
  BALANCE=BALANCE*1.10 ENDIF

SORT NAME 100 USING ACCOUNT UNTIL ARFILE  <----

LIST ACCOUNT NAME BALANCE
T1 'REPORT ACCOUNT WITH NAME' WITH 2 AFTER
```

Figure 160 SORT Statement [Example 11](#)

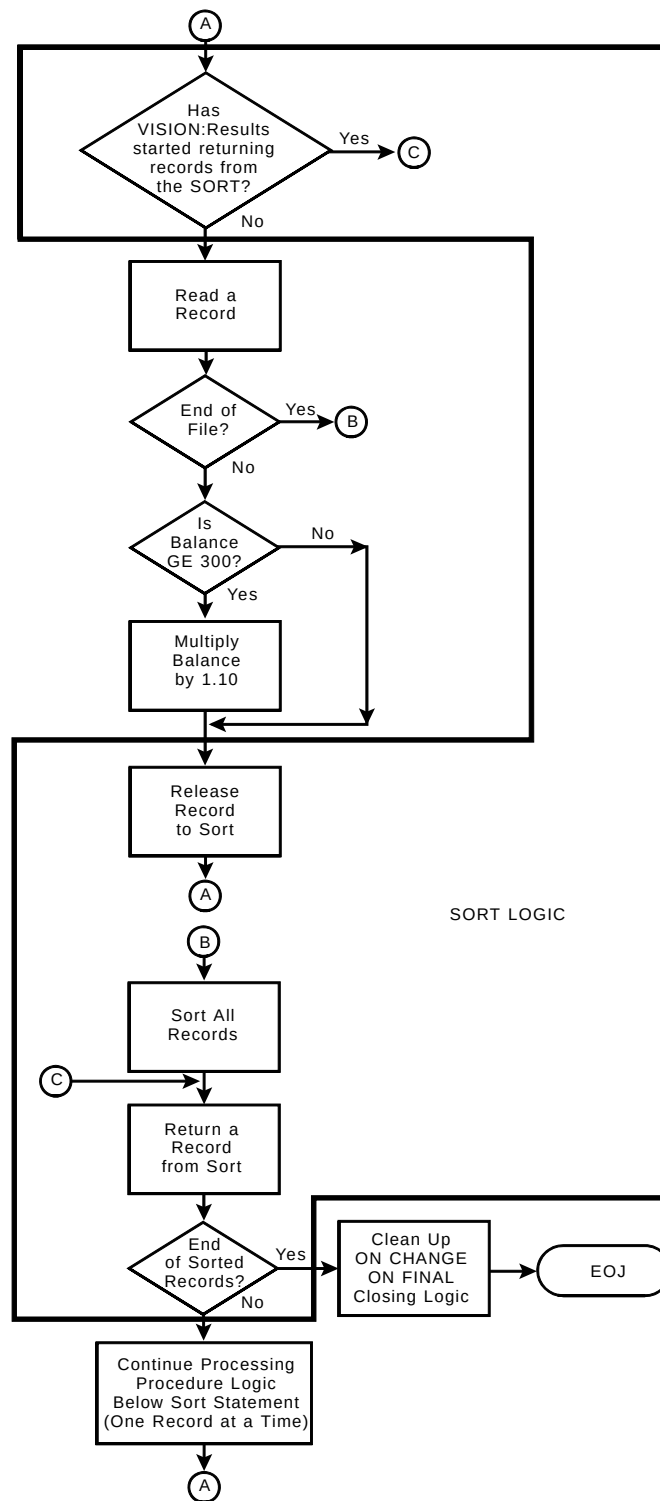


Figure 161 SORT UNTIL Filename Flow Diagram

Example 12

The following example demonstrates merging partial records from two files and terminating the input phase of the sort when end of file is detected on the second file.

```
FILE ARQ1FIL FB 352 5280 STATUS ARQ1EOF
  ARQ1REC 352 NAME 25 85
    BALANCE 5 170 PD 2 A ACCOUNT 2 182
FILE ARQ2FIL FB 352 10560
  ARQ2REC 352

READ ARQ1FIL
IF ARQ1EOF EQ 'E' GOTO READQ2 ENDIF
GOTO SORTREC
READQ2:
  READ ARQ2FIL
  MOVE ARQ2REC TO ARQ1REC
SORTREC:
  SORT NAME 100 USING ACCOUNT UNTIL ARQ2FIL  <----
  LIST ACCOUNT BALANCE
T1 'REPORT MERGE' WITH 2 AFTER
```

Figure 162 SORT Statement [Example 12](#)

[Figure 163](#) illustrates the logic, both user and internal.

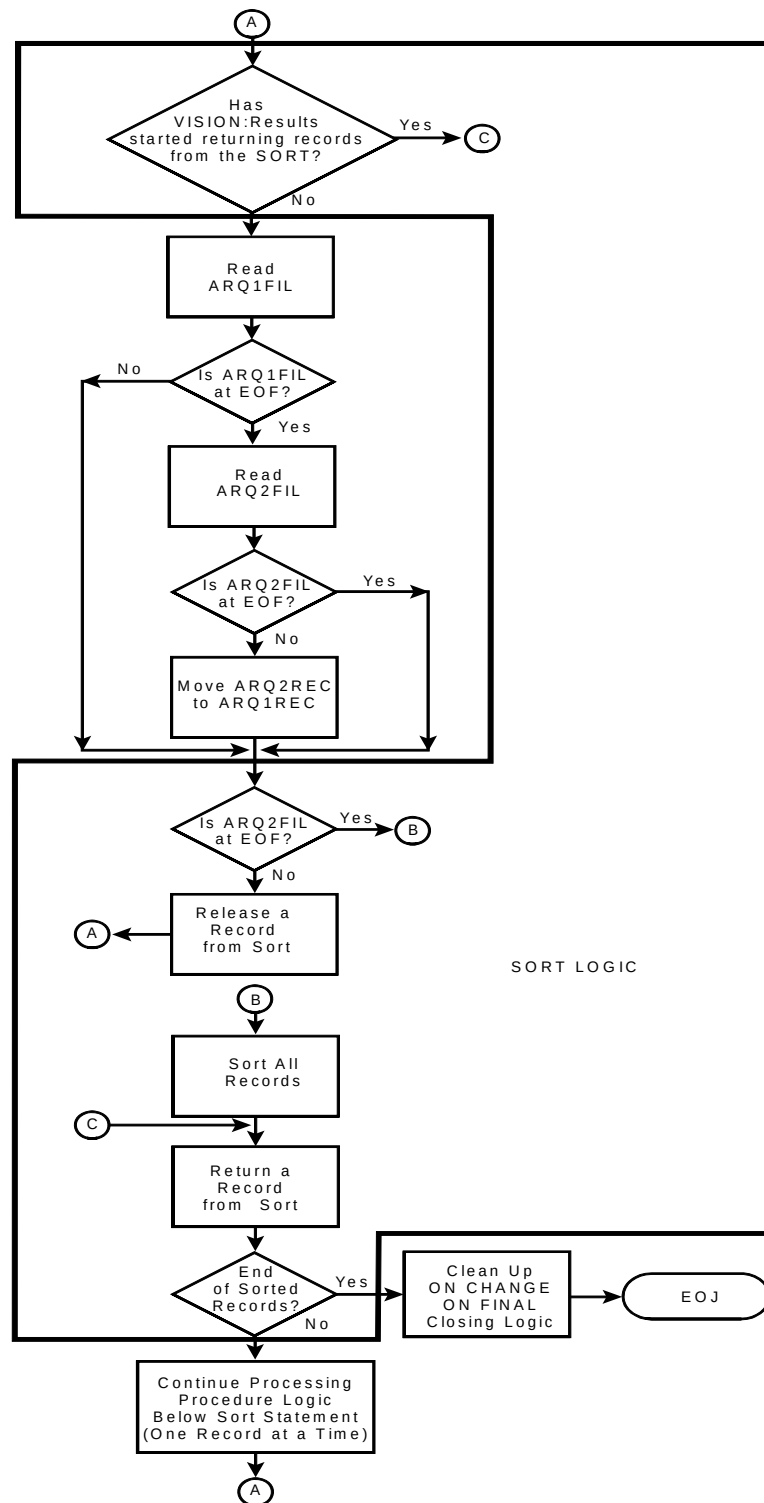


Figure 163 Merge Partial Records from Two Files with SORT Flow Diagram

UNTIL dataname

If the data name used in the UNTIL of a SORT is associated with the STATUS keyword ([Example 13](#)) of a file and that file goes to end of file, it automatically is set to E by VISION:Results, therefore terminating the input phase of the SORT.

Example 13

```
FILE ARFILE FB 352 5280 STATUS EOFEQE
  ACCOUNT 2 182
  BALANCE 5 170 PD 2 A
WORKAREA
  WBALANCE 5 PD 2 E (BALANCE'AMOUNT)
  WACCOUNT 2 (ACCOUNT'NUMBER)
  WDESC 22 (REMARKS)
READ ARFILE
IF ACCOUNT EQ 'WO'
  MOVE 'E' TO EOFEQE GOTO SORTREC ENDIF
MOVE ACCOUNT TO WACCOUNT
MOVE BALANCE TO WBALANCE
MOVE SPACES TO WDESC
IF BALANCE LE 0 MOVE 'BALANCE TO BE EXAMINED' TO
  WDESC GOTO SORTREC ENDIF
SORTREC:
  SORT WBALANCE 29 USING WACCOUNT UNTIL EOFEQE
  LIST WACCOUNT WBALANCE WDESC
  T1 'REPORT 1' WITH 2 AFTER
```

Figure 164 SORT Statement [Example 13](#)

RELSORT and RETSORT Commands

The RELSORT and RETSORT commands alleviate the programming difficulties that occur when using the SORT command in a sophisticated VISION:Results program. The RELSORT command releases a record to SORT and then the logic flows directly to the next instruction below the RELSORT command. The RETSORT command retrieves a record from SORT and then the logic flows directly to the next instruction below the RETSORT command.

RELSORT

Figure 165 RELSORT Command

RETSORT

Figure 166 RETSORT Command

One or more RELSORT and RETSORT commands can be included in the VISION:Results program. The RELSORT and RETSORT commands can be used subordinate to an IF statement. They can be included within performed logic, and they can be used in conjunction with executing the actual SORT command to release and retrieve sorted records. However, if you use a mixture of SORT, RELSORT, and RETSORT commands to handle your sorted records, you must be thoroughly familiar with the rules governing these commands.

The rules and considerations for using the RELSORT and RETSORT commands are:

- Only one SORT command can be included in the program. The SORT command can be placed in a non-executable place within the program.
- The output (retrieval) phase of SORT is initiated the first time any of the following conditions occur:
 - A RETSORT command is executed.
 - A read is issued (either explicitly using a READ command or implicitly by VISION:Results automatic cycle) in which end of file occurs and the format of the SORT statement is either:

```
SORT filename USING field1 field2 ...
```

or

```
SORT filename USING field1 field2 ...  
                UNTIL filename
```

In this case, immediately after the detection of the end of file occurs on the file being sorted or the file used in the UNTIL keyword, the flow of logic transfers directly to the SORT command and the first record is retrieved from SORT. The logic then flows to the statement directly below the SORT command.

- The SORT command is executed; the format of the SORT command is either:

```
SORT filename USING field1 field2 ...  
                UNTIL dataname
```

or

```
SORT workarea USING field1 field2 ...  
                UNTIL dataname
```

The UNTIL data name contains the value E. In this case, when the SORT command is executed, a record is retrieved from SORT and then the flow of logic is to the statement directly below the SORT command.

- Once the output (retrieval) phase of SORT is initiated, the RELSORT command should not be executed. Any attempt to do so results in a 500 code abend.
- The RETSORT command cannot be issued in ON END OF SORTING logic. Any attempt to do so results in a 500 code abend.

Example 1 Accounts Receivable File Sorted into Two Distinct Groups

The first group represents all MO accounts. The second group represents all accounts with a balance greater than \$100.00. All other records are not included in the sort. It is possible that there are records that are MO with a balance greater than \$100.00. In these cases, the record is passed to sort twice: once for the MO group and once for the over \$100.00 group.

Program Narrative

The record to sort is defined in a work area as shown in Statements 8 through 13. The MO group is given a group code of 1 while the over \$100.00 group is given a group code of 2. The records are sorted by group code, account code, and account number, as shown in Statement 33. UNTIL ARFEND is included because the record to be sorted is in a work area. The ARFEND data name is the STATUS data name for the input file and consequently contains the value E when the file is at end of file. The testing for the value E in ARFEND is automatically done at the time the SORT command is executed. This condition initiates the retrieval phase of SORT. This means that the program must branch to the SORT command when the input file comes to end of file as shown in Statement 16; otherwise, the program loops endlessly, never entering the SORT retrieval phase.

The MO records are formatted and released to SORT using the RELSORT command (Statement 23) as shown in Statements 18 through 23. The over \$100.00 group is formatted and released to SORT by falling through and executing the SORT command as shown in Statements 26 through 33.

The retrieval logic is represented by the statements below the SORT command exactly as the SORT command normally works. Statements 35 and 36 are executed for each record retrieved from SORT.

Program Listing

```

1  FILE ARFILE FB 352 5280 STATUS ARFEND
2  ACCTNO      7  4
3  NAME       25 85
4  BALANCE    5 170 PD  2
5  ACCTCODE   2 182
6
7  WORKAREA
8  WKREC      40  1
9  WGROUP     1  1      (GROUP)
10 WACCTCODE  2      (ACCOUNT' CODE)
11 WACCOUNT   7      (ACCOUNT' NO.)
12 WNAME     25      (NAME)
13 WBALANCE   5 PD  2 E (BALANCE)
14
15 CONTROL WGROUP
16 IF ARFEND EQ 'E' GOTO SORTCMD ENDIF
17 IF ACCTCODE EQ 'MO'
18   MOVE '1' TO WGROUP
19   MOVE ACCTCODE TO WACCTCODE
20   MOVE ACCTNO TO WACCOUNT
21   MOVE NAME TO WNAME
22   MOVE BALANCE TO WBALANCE
23   RELSORT
24   ; <----
25 ENDIF
26 IF BALANCE LT 100.00 REJECT ENDIF
27   MOVE '2' TO WGROUP
28   MOVE ACCTCODE TO WACCTCODE
29   MOVE ACCTNO TO WACCOUNT
30   MOVE NAME TO WNAME
31   MOVE BALANCE TO WBALANCE
32
33 SORTCMD:
34   SORT WKREC USING WGROUP WACCTCODE WACCOUNT UNTIL ARFEND

```

Figure 167 Program Listing (Page 1 of 2)

```

35 LIST SUPPRESS (WGROUP WACCTCODE) WACCOUNT WNAME
36 WBALANCE
37 ACCEPT
38 ON CHANGE IN WGROUP
39 LIST ' ' WITH EJECT AFTER
40 T1 'VISION:RESULTS REPORT 1' WITH 2 AFTER
41 T1+30 DYLDATE
42 T1+90 DYLPAGE

```

Figure 167 Program Listing (Page 2 of 2)

Sysout

```

01/15/99          VISION:RESULTS REPORT 1          PAGE    1
GROUP    ACCOUNT  ACCOUNT      NAME              BALANCE
          CODE      NO.
1         MO      2002922  FLOWERS, ETHEL      2,541.70 <----
                  6001327  CARLON, MR MARIANO    118.00 <----
                  6007724  HARRINGTON, RUTH      37.60
                  6012132  ZENZOLA, MICHAEL      30.20
                  6017479  BERTSON, ANNA         3.00

```

Figure 168 System Output for Listing Page 1

```

01/15/99          VISION:RESULTS REPORT 1          PAGE    2
GROUP    ACCOUNT  ACCOUNT      NAME              BALANCE
          CODE      NO.
2         FO      8011508  HUGHES, RAY         178.70
                  2002299  PLACIDO, ORTEGA       413.58
                  6218113  AGUIERA, EMILIO       108.44
                  KO      1014021  WOOD, ROGER D         1,702.42
                  2005131  WICINSKI, ALEXANDER    848.71
                  6088678  TYLER, JAMES          305.64
                  7014996  OLVERA, CLEMENTINE    106.74
                  7072694  WARTON, LEE           224.00
                  MF      1013904  HINDERLICH, PAUL      1,142.31
                  1014889  SULLIVAN, JOSEPH C     337.70
                  4003896  PETERSON, WALTER      196.00
                  MI      2004739  STEIDLEY, GURTHA N     136.23
                  6008321  YEAGER, CLIFFORD M     140.00
                  MN      2005956  SOTO, RUFINO          125.29
                  MO      2002922  FLOWERS, ETHEL      2,541.70 <----
                  6001327  CARLON, MR MARIANO    118.00 <----
                  MT      2014017  HERNANDEZ, CARLOS     258.00
                  MU      1014706  HOLLENBECK, HOME TR   406.46
                  6105904  CRESS, NELLIE         176.00

```

Figure 169 System Output for Listing Page 2

Example 2 Master File to be Updated

A transaction file contains the information that is to be used to update the master file. However, the transaction file must first be sorted into key sequence. The transaction file can contain more than one record for a given key.

Program Narrative

This program is an example of using the RELSORT and RETSORT commands without actually executing the SORT command. Therefore, the SORT command is in a never-executed position (Statement 76) within the VISION:Results program. The program consists of a very short main-line piece of logic (Statements 23 through 25) that performs the input-to-sort logic and then performs the output-to-sort logic, and finally issues a STOP command. The input-to-sort logic (Statements 27 through 33) reads a record from the input file. If the record is an A type record, it is passed to SORT using the RELSORT command. When the input file comes to end, the performed logic is exited.

The output-to-sort logic (Statements 35 through 66) is composed of retrieving transaction records from SORT using the RETSORT command, reading the master file, and keeping both files in sync on the key field. When the first execution of the RETSORT command occurs (Statement 36), the SORT automatically enters its output (retrieval) phase. The ON END OF SORTING logic (Statements 78 through 80) moves high-values into the transaction key, so the match/merge logic continues to operate correctly when the branch back into the performed logic is taken (GOTO OUTSORT2).

Program Listing

```

1  FILE TRANFILE FB 80 STATUS TRANEND
2  TRANKEY      3      1
3  TRANCODE     1      4
4  TRANFLDS    76      5
5  TRANFLD1     5      5
6  TRANFLD2     4
7
8  FILE MSTRFILE FB 80 STATUS MSTREND
9  MSTRREC      80      1
10 MSTRKEY       3      1
11 MSTRFLDS     76      4
12 MSTRFLD1      5      4
13 MSTRFLD2      4
14
15 FILE OUTMSTR FB 80 OUTPUT FROM OUTMSTR
16 OUTREC       80      1
17 OUTKEY        3      1
18 OUTFLDS      76      4
19 OUTFLD1       5      4
20 OUTFLD2       4
21
22 MAIN:
23   PERFORM INPSORT TO INPSORTX
24   PERFORM OUTSORT TO OUTSORTX
25   STOP
26
27 INPSORT:
28   READ TRANFILE
29   IF TRANEND EQ 'E' GOTO INPSORTX ENDIF
30   IF TRANCODE EQ 'A' RELSORT ENDIF      ; <----
31   GOTO INPSORT
32
33 INPSORTX:
34
35 OUTSORT:
36   RETSORT                                     ; <----
37 OUTSORT1:
38   READ MSTRFILE
39   IF MSTREND EQ 'E'
40     AND TRANKEY EQ X'FFFFFF'
```

Figure 170 Program Listing (Page 1 of 2)

```

41      STOP
42    ENDIF
43  OUTSORT2:
44    IF MSTRKEY LT TRANKEY
45      MOVE MSTRREC TO OUTREC
46      WRITE OUTMSTR
47      GOTO OUTSORT1
48    ENDIF
49    IF MSTRKEY GT TRANKEY
50      MOVE TRANKEY TO OUTKEY
51      MOVE TRANFLDS TO OUTFLDS
52      WRITE OUTMSTR
53      RETSORT                                ; <----
54      GOTO OUTSORT2
55    ENDIF
56    IF MSTRKEY EQ X'FFFFFF'
57      GOTO OUTSORTX
58    ENDIF
59  OUTSORT3:
60    IF MSTRKEY NE TRANKEY
61      MOVE MSTRREC TO OUTREC
62      WRITE OUTMSTR
63      READ MSTRFILE
64      GOTO OUTSORT2
65    ENDIF
66    PERFORM UPDATEREC TO UPDATERECX
67    RETSORT                                ; <----
68    GOTO OUTSORT3
69
70  OUTSORTX:
71
72  UPDATEREC:
73    MOVE TRANKEY TO OUTKEY
74    MOVE TRANFLDS TO OUTFLDS
75  *      .
76  *      .      .--- ADDITIONAL UPDATE LOGIC
77  *      .
78  UPDATERECX:
79
80  SORT TRANFILE USING TRANKEY UNTIL TRANEND ; <----
81
82  ON END OF SORTING
83    MOVE X'FFFFFF' TO TRANKEY
84    GOTO OUTSORT2

```

Figure 170 Program Listing (Page 2 of 2)

Limitations and Constraints

- Only one SORT command is allowed per VISION:Results program request.
For z/OS only, if the record length is not coded on the FILE statement, VISION:Results accesses the information from the JCL or file label at compile time and places the record length into the generated SORT control statement. Otherwise, the record length *must* be coded on the FILE statement.
- The SORT command cannot be subordinate to an IF statement.

```
IF ACCOUNT EQ 'WO' REJECT ENDIF
SORT ARFILE USING ACCOUNT
```
- The SORT command cannot be subordinate to an ON ONE command.
- A file that is USED in a secondary request cannot be specified as the operand of the UNTIL filename on a SORT statement.

```
USE ARFILE
```

- The SORT command can be used only at detail time processing. It cannot be subordinate to an ON CHANGE, ON FINAL, ON END OF SORTING, or ON END OF INPUT statement.
- The SORT command and the ON END OF INPUT command are mutually exclusive. They cannot be used in the same program. SORT users must use the ON END OF SORTING command.
- When using the ON END OF SORTING command, a STOP command must be issued.
- The ON END OF SORTING command must precede any ON CHANGE IN or ON FINAL commands.
- Never use the ACCEPT or REJECT commands subordinate to ON END OF SORTING.
- If trailer records, and so on, are generated at ON END OF SORTING, use the WRITE command throughout your procedure logic.
- Because ON END OF SORTING is a detail-time process, the SUM, SUMn, or PRIOR of any field is prohibited.
- Do not use SORT within a STOP statement in a VISION:Results program. Because of the structure of the SORT facility, unpredictable results can occur when a SORT command is performed. Use the RELSORT and RETSORT commands instead.
- A sort key cannot be greater than 255 bytes in length.
- The starting location of a sort key must be within the first 4092 bytes of the record or work area being sorted.

Problem Areas

A return code of 500 is issued by VISION:Results when a non-zero return code has been returned from your installation's sort/merge program. Generally this occurs when large records being sorted have exceeded available memory, there is not enough memory available to begin with, or insufficient space has been allocated for the sort work files. If this occurs, take one of the following actions.

z/OS

- VISION:Results for z/OS defaults to a minimum of 36K and a maximum of 100K for sort work memory. These values can be overridden by the DYLINSTL parameter SORTMEM=(minimum, maximum). For more information about the DYLINSTL macro, see the *Advantage VISION:Results for z/OS Installation Guide*.
- When using SORT, increase the region size to at least 250K to maximize full memory utilization.
- Add or increase the nK keyword on your SORT statement. A 300K region is required for this example:

```
SORT ARFILE 200K USING ACCOUNT TRANS
```

- Increase the number of SORT work files (JCL for z/OS).

VSE

- VISION:Results for VSE defaults to the available unused partition space for sort work memory.
- Increase the partition size from 200K to 250K. Virtual users should increase the SIZE parameter on the EXEC statement (for example, SIZE=250K). Increase the number of sort work files (JCL and WORK n for VSE).

An abend of 4001 occurs when the user re-enters the SORT after all sorted records have been exhausted.

z/OS and VSE

Specify ON END OF SORTING and set a status indicator to E. When VISION:Results senses that all sorted records have been exhausted, it enters this logic, setting the status indicator. In your program logic, an IF statement can be used to check the status indicator and take the necessary action. [Example 10 on page 255](#) demonstrates this technique.

Under z/OS, you can periodically run into an 0C4 or 0C5 abend because the sort has not been completed (that is, not all the records were returned from the SORT before a STOP was executed). This condition normally occurs when two or more files are being processed in the same program. [Example 4 on page 245](#) shows you how to prevent this condition. To prematurely end the output phase of the SORT, see [Example 10 on page 255](#).

Sort Considerations

VSE

- The sort facility of VISION:Results starts the sort program with a LOAD instruction using the program name SORT. If the sort/merge program is named something other than SORT, indicate the name using the DYLINSTL parameter SORTNAM when creating your customizing module. For information about the DYLINSTL macro, see the *Advantage VISION:Results for z/OS Installation Guide*.
- Because of the combined memory requirements of VISION:Results and SORT, conserve memory as much as possible. If memory is limited, your VISION:Results input and output FILE statements should indicate single buffering (ONEBUF).

z/OS

- The sort facility of VISION:Results starts the sort program with LINK using the program name SORT. If the sort/merge program is other than SORT, indicate the name using the DYLINSTL parameter SORTNAM when creating your customizing module. For information about the DYLINSTL macro, see the *Advantage VISION:Results for z/OS Installation Guide*.
- Because of the combined memory requirements of VISION:Results and SORT, conserve memory as much as possible. If memory is limited, your VISION:Results input and output file DD statements should indicate single buffering in the DCB parameter (DCB=BUFNO=1).

z/OS JCL Requirements

The z/OS JCL requires SORTLIB, SYSOUT, and SORTWK01-nn DD statements in addition to those required by VISION:Results and your FILE definitions. SORTIN and SORTOUT DD statements are not needed.

DISK input and output	//RESULTS	JOB	(accounting information...)
	//STEP01	EXEC	PGM=DYL280,REGION=250K,TIME=2
	//STEPLIB	DD	DSN=your.load.library,DISP=SHR
	//SYSCOPY	DD	DSN=your.copylib,DISP=SHR
	//SYS280R	DD	SYSOUT=A
	//SYSPRINT	DD	SYSOUT=A
	//AUDPRINT	DD	SYSOUT=A
	//AUDWORK	DD	UNIT=SYSDA,SPACE=(TRK,(10,5))
	//AUDEPF	DD	UNIT=SYSDA,SPACE=(TRK,(10,5)),
	//		DCB=(BLKSIZE=800,LRECL=80,RECFM=FB)
Required only if using the letter writing, sampling, or regression functions	//AUDCBF	DD	UNIT=SYSDA,SPACE=(TRK,(10,5)),
	//		DCB=BLKSIZE=1000
	//SYSOUT	DD	SYSOUT=A
	//SORTLIB	DD	DSN=your.sortlib.dataset,DISP=SHR
	//SORTWK01	DD	UNIT=SYSDA,SPACE=(CYL,(5),,CONTIG)
	//SORTWK02	DD	UNIT=SYSDA,SPACE=(CYL,(5),,CONTIG)
	//SORTWK03	DD	UNIT=SYSDA,SPACE=(CYL,(5),,CONTIG)
	//FILEA	DD	DSN=your.VISION:Results.demofile,DISP=SHR
	//FILEB	DD	DSN=your.output.file,DISP=(,PASS),
Required only if SORT is used	//		UNIT=SYSDA,SPACE=(TRK,(5,5))
	//SYSIN	DD	*
			your VISION:Results program
	//*		
Input file JCL if DISK	//		

Output file JCL if DISK

Figure 171 z/OS JCL Requirements

VSE JCL Requirements

The VSE JCL requires SORTWK1-n DLBL/EXTENT statements in addition to those required by VISION:Results. The SYS number begins with SYS001. SORTIN and SORTOUT DLBL/EXTENT statements are not required.


```

// JOB      RESULTS
// ASSGN SYS012,X'cuu'
// DLBL FILEA,'your.input.file'
// EXTENT SYS012,...extent information

// ASSGN SYS021,X'cuu'
// DLBL FILEB,'your.output.file'
// EXTENT SYS021,...extent information

// ASSGN SYS004,X'cuu'
// DLBL IJSYS04,'VISION:RESULTS WORK FILE',0
// EXTENT SYS004,111111,1,0,20,20

// ASSGN SYS008,X'cuu'
// DLBL SYS008,'WORK FILE2',0
// EXTENT,SYS008,...extent information

// ASSGN SYS009,X'cuu'
// DLBL SYS009,'LETTERS ETC.SYSLST',0
// EXTENT,SYS009,...extent information

// ASSGN SYS010,X'cuu'
// DLBL SYS010,'WORK FILE3',0
// EXTENT,SYS010,...extent information

// ASSGN SYS011,X'cuu'
// DLBL SYS011,'WORK FILE4',0
// EXTENT,SYS011,...extent information

// ASSGN SYS001,X'cuu'
// ASSGN SYS002,X'cuu'
// ASSGN SYS003,X'cuu'
// DLBL SORTWK1,'VISION:RESULTS SORT WORK 1',0
// EXTENT SYS001,...extent information
// DLBL SORTWK2,'VISION:RESULTS SORT WORK 2',0
// EXTENT SYS002,...extent information
// DLBL SORTWK3,'VISION:RESULTS SORT WORK 3',0
// EXTENT SYS003,...extent information

// EXEC DYL280,SIZE=250K
OPTION DISK dddd *

your VISION:Results program
/*
/ &

```

INPUT file JCL if
DISK

Output file JCL

Assign only if
IJSYS04 is not a
standard label file

Bold text is
required only if
using the letter
writing, sampling,
or regression
functions

Required only if
SORT is used

'cuu' refers to the
device address.

Figure 172 VSE JCL Requirements

CMS FILEDEF Requirements

Tells CMS that a disk file with the file name DYLDATA and the file type DATA is referred to as INFILE in your Vision:Results program. In this example, this is your input data file.

Tells CMS that a disk file with the file name DYLPROG and the file type DATA is referred to as SYSIN by Vision:Results. This data set contains your program statements.

```

ERASE SYSPRINT LISTING
ERASE AUDPRINT LISTING
ERASE SYS280R LISTING
ERASE SYS004 DATA
ERASE AUDWORK DATA
ERASE AUDCBF DATA
ERASE AUDEPF DATA

```

```

FILEDEF SYSPRINT DISK SYSPRINT LISTING A (DISP MOD)
FILEDEF AUDPRINT DISK AUDPRINT LISTING A (DISP MOD)
FILEDEF SYS280R DISK SYS280R LISTING A

```

```

FILEDEF SYS004 DISK SYS004 DATA A
FILEDEF AUDWORK DISK AUDWORK DATA A
FILEDEF AUDCBF DISK AUDCBF DATA A
FILEDEF AUDEPF DISK AUDEPF DATA A
FILEDEF INFILE DISK DYLDATA DATA A
FILEDEF SYSIN DISK DYLPROG DATA A
DYL280

```

Figure 173 CMS FILEDEF Requirements

Note: AUDPRINT, AUDWORK, AUDCEF, and AUDEPF file definitions are required only when using the letter writing, sampling, and regression functions.

Chapter 13: Using the CALL Command

VISION:Results allows external subroutines and subprograms to be started using the CALL command. In this chapter, the term *subroutine* refers to those external routines COBOL programmers often called subprograms. External subroutines can be written in COBOL, FORTRAN, PL/I, or Assembler language. The function of the external subroutine is dependent upon the particular application. Special input and output considerations, database files, and Julian to Gregorian date conversion are just a few of the functions that can be handled by an external subroutine.

The CALL statement indicates the external subroutine to be started and, optionally, a parameter list of data areas (data names) required by the called subroutine.

The format of the CALL command is:

```
CALL subroutine name [nK] [CDLOAD | NOCDLOAD] [USING |dataname|literal|  
D'dataname|E'dataname|L'dataname|O'dataname|P'dataname| T'dataname]
```

Figure 174 CALL Command Format

Subroutine Name

The subroutine name immediately follows the CALL command. This is the phase name of the subroutine in VSE; in z/OS, it is the member name of the subroutine in a load library.

nK

Note: This section only applies to VSE.

If CDLOAD is coded on the CALL statement, or was specified on the OPTION statement or in the DYLINSTL macro, the nK (size) parameter is not required. If CDLOAD is not in effect, the following applies:

- The VSE user additionally is required to supply the memory requirements for the subroutine in nK, where nK is in 1K increments from 1 to 999.
- VISION:Results defaults to 4K on the CALL statement. Use the default unless memory constraints require that you specify a smaller memory size.

CDLOAD, NOCDLOAD

Note: This section only applies to VSE.

The CDLOAD parameter causes the named subroutine to be loaded using the VSE CDLOAD facility. The subroutine is loaded into the partition's dynamic area, rather than into storage reserved by the compiler. Because of this, the size parameter on the CALL is not needed—VSE determines the storage required.

If the size parameter was previously used to reserve additional storage at the end of the phase for the subroutine's use, this storage is no longer available. In this case, the CALL should specify NOCDLOAD and use the size value as before. CDLOAD or NOCDLOAD on the CALL statement overrides any OPTION statement or DYLINSTL specification.

To further assist you, a table containing names of phases that should not be loaded with CDLOAD has been provided. VISXLOAD is a phase that can be modified by your installation by adding any phase names that must not be loaded using CDLOAD. Two macros, VISXLOAD and VISXEND, have been provided for this purpose and were added to the same library containing the DYLINSTL macro during the compiler's installation. Use the following example to add new names to the VISXLOAD table.

```
// JOB      (installation standard job statement)
```

For an example of JCL used to assemble and link edit a phase, see the *Advantage VISION:Results for VSE Installation Guide*.

```
// OPTION  CATAL
// PHASE   VISXLOAD,0
// EXEC   ASSEMBLY
//          COPY VISXLOAD
//          COPY VISXEND
//          VISXLOAD
//          VISXLOAD PHASE=sub1
//          VISXLOAD PHASE=sub2
//          .
//          .
//          VISXEND
//          END
/*
// EXEC   LNKEDT
/END
```

In this example, subroutines sub1 and sub2 are excluded from the CDLOAD process. A subroutine's name in the VISXLOAD table prevents it from being included in the CDLOAD process and overrides any CDLOAD specification in the CALL or OPTION statements or the DYLINSTL macro.

USING

The keyword USING, followed by the parameter list, is optional and need not be used if the called subroutine does not require a parameter list. If a parameter list is required, the data names must be separated by spaces. The data names must be coded in the order expected by the subroutine.

Parameters

Your system is the only limit to the number of parameters that can be passed.

```
CALL CHECKPT
CALL CHECKR 6K VSE
CALL CHECKIT USING FIELD1 FIELD2 OS/390
CALL CHECKIT CDLOAD USING FIELD1 FIELD2 VSE
CALL CHECKIT 1K USING FIELD1, FIELD2, FIELD3, OS/390 & VSE
FIELD4, FIELD5, FIELD6,
FIELD7
CALL DATBAS USING FIELD1 FIELD2 FIELD3 FIELD4 OS/390
FIELD5 FIELD6 FIELD7
CALL RUNSUB USING DATAFLD1 'LITERAL 1' DATAFLD2 L'DATAFLD3
```

Attributes of data fields can be passed to a subroutine. This is achieved by using a parameter composed of the data field name preceded by an apostrophe and a specific letter. The special attributes are as follows:

dataname	The dataname can be from 2 to 50 alphanumeric characters and must start with an alphabetic character. The installation default is 50 characters.
literal	A character or alphanumeric value expressed by enclosing it in single or double quotation marks. The parameter passed consists of the literal, preceded by a 2-byte binary field containing the length of the literal.
D'dataname	The parameter passed is 2 bytes—the letter D followed by a 1-byte zoned-decimal number that represents the number of digits to the right of the assumed decimal point in the data name field.
E'dataname	The parameter passed is 2 bytes—the letter E followed by the letter (E, A, Z, B, P, D, S, and so on) that represents how the data name field is to be edited when listed in the report.
L'dataname	The parameter passed is 3 bytes—the letter L followed by a 2-byte binary number that represents the length of the data name field.
O'dataname	The parameter passed is 3 bytes—the letter O followed by a 1-byte zoned-decimal number that represents the number of digits to the right of the printed decimal point when the data name field is listed in the report. If the number of digits to print to the right of the decimal point is not specified, the 1-byte number is blank.

P'dataname	The parameter passed is 4 bytes—the letter P followed by a 3-byte zoned-decimal number that represents the size of the print field when the data name field is listed in the report. If the print field size is not specified, the 3-byte number is blank.
T'dataname	The parameter passed is 2 bytes—the letter T followed by a letter (N for numeric, C for character, P for packed, and B for binary) that describes the data name field type.

Return Code RETCODE

You can gain accessibility to the return code RETCODE (in register 15) from a called subroutine. In the DYLINSTL macro, code:

```
RETCODE=Y
```

In addition, include the following program logic:

```
WORKAREA
  XX      2
  REDEFINE XX
  YY      2  BI
CALL subroutine USING parameters
MOVE DYLCOMRG TO XX
IF YY NE 0
  handle non-zero return-code
ENDIF
```

For more information on the DYLINSTL macro, see the *Advantage VISION:Results for z/OS Installation Guide*.

External Subroutine Considerations

The link set up by VISION:Results to an external subroutine conforms to COBOL, Assembler, PL/1, and FORTRAN conventions. See subsequent sections for a more information about linkage conventions.

External subroutines must be link edited into a Phase/Core Image Library (VSE) or a load library (z/OS). Under z/OS, the load library must be defined by a JOBLIB or STEPLIB DD statement if it is not a library that is automatically referenced by the system. Under VSE, external subroutines to be invoked by VISION:Results must be relocated dynamically by a special routine or loader.

During the initialization phase, all subroutines named in CALL statements are loaded into memory. During the execution phase, they are entered whenever the associated CALL statement is encountered. Only one copy of an external subroutine is ever loaded even though it can appear in more than one CALL statement.

Multiple entry points are not supported, so the external subroutine must have only one entry point.

Language Environment Considerations

You can CALL subroutines written in any LE-compliant language, including COBOL, C, PL/I, FORTRAN, and Assembler. You can also CALL the IBM LE Service Routines, such as CEEDATE and CEEDAYS.

You can request this as a default by coding the DYLINSTL parameter LE=Y, which overrides the DYLINSTL parameter COBENV=Y. If both are specified, VISION:Results issues a warning message which states that the COBENV option will be ignored.

You can specify or override this for a single run by specifying LE | NOLE on the OPTION statement.

```
OPTION [LE | NOLE]
```

Figure 175 LE or NOLE

OPTION LE causes VISION:Results to establish the LE environment even if the DYLINSTL macro did not specify LE=Y. OPTION NOLE prevents VISION:Results from establishing the LE environment even if the DYLINSTL macro did specify LE=Y. The options can be specified with OPTION FREEZE and OPTION RESTORE; however, the RESTORE with LE only works if the program was frozen with VISION:Results r5 or later.

For more information on the DYLINSTL macro, see the *Advantage VISION:Results for z/OS Installation Guide*.

Database Access

Called subroutines can access IMS/DB or DL/I databases if the necessary coding changes have been made to the called programs and either of the following interfaces is active:

- VISION:Results Interface to IMS/DB z/OS
- VISION:Results Interface to DL/I VSE

Note: Access to either the IMS/DB or DL/I databases is available only to licensed users of either VISION:Results Interface to IMS/DB z/OS or VISION:Results Interface to DL/I VSE.

Access to the IMS/DB or DL/I databases is provided through the reserved word, DYLSYSPARM. DYLSYSPARM contains the address of the list of the associated Program Communication Blocks (PCBs) reflected in the Program Specification Block (PSB). The list of the associated PCBs has the addresses of the I/O and DB PCBs required to directly access the IMS/DB or DL/I databases. The CALL statement must have DYLSYSPARM as the first parameter, as follows:

```
CALL subroutine USING DYLSYSPARM
```

The PSB pointer is not provided within DYLSYSPARM only if one of the following occurs:

- More than 48 characters are being passed from PARM=UA in the JCL for (z/OS)
- JCL contains the // OPTION SYSPARM statement (VSE)

The called program must resolve the addresses of the list of associated PCBs to the PCB masks within the program in order to access the IMS/DB or DL/I databases. To accomplish this, code the following LINKAGE SECTION in a COBOL program. This COBOL example assumes 4 DB PCBs reside within the PSB.

```
LINKAGE SECTION.
77  PCBLIST-PTR          USAGE POINTER.
01  PCB-PTRS.
    02  PCB1-PTR          USAGE POINTER.
    02  PCB2-PTR          USAGE POINTER.
    02  PCB3-PTR          USAGE POINTER.
    02  PCB4-PTR          USAGE POINTER.
01  DB-PCB1.
    02  PCB1-DBD-NAME     PIC X(08).
    02  PCB1-SEG-LEVEL    PIC X(02).
    02  PCB1-STAT-CODE    PIC X(02).
    02  PCB1-PROC-OPT     PIC X(04).
    02  FILLER            PIC S9(5) COMPUTATIONAL.
    02  PCB1-SEG-NAME     PIC X(08).
    02  PCB1-LEN-KFB      PIC S9(5) COMPUTATIONAL.
    02  PCB1-NU-SENSESEG  PIC S9(5) COMPUTATIONAL.
    02  PCB1-KEY-FB       PIC X(??).
01  DB-PCB2.
    02  PCB2-DBD-NAME     PIC X(08).
    02  PCB2-SEG-LEVEL    PIC X(02).
    02  PCB2-STAT-CODE    PIC X(02).
    02  PCB2-PROC-OPT     PIC X(04).
    02  FILLER            PIC S9(5) COMPUTATIONAL.
    02  PCB2-SEG-NAME     PIC X(08).
    02  PCB2-LEN-KFB      PIC S9(5) COMPUTATIONAL.
    02  PCB2-NU-SENSESEG  PIC S9(5) COMPUTATIONAL.
    02  PCB2-KEY-FB       PIC X(??).
01  DB-PCB3.
    02  PCB3-DBD-NAME     PIC X(08).
    02  PCB3-SEG-LEVEL    PIC X(02).
    02  PCB3-STAT-CODE    PIC X(02).
    02  PCB3-PROC-OPT     PIC X(04).
    02  FILLER            PIC S9(5) COMPUTATIONAL.
    02  PCB3-SEG-NAME     PIC X(08).
    02  PCB3-LEN-KFB      PIC S9(5) COMPUTATIONAL.
    02  PCB3-NU-SENSESEG  PIC S9(5) COMPUTATIONAL.
    02  PCB3-KEY-FB       PIC X(??).
01  DB-PCB4.
```

Figure 176 Sample COBOL Program Linkage Section to Access IMS/DB or DL/I Databases (Page 1 of 2)


```

02 PCB4-DBD-NAME      PIC X(08) .
02 PCB4-SEG-LEVEL     PIC X(02) .
02 PCB4-STAT-CODE     PIC X(02) .
02 PCB4-PROC-OPT      PIC X(04) .
02 FILLER             PIC S9(5) COMPUTATIONAL.
02 PCB4-SEG-NAME      PIC X(08) .
02 PCB4-LEN-KFB       PIC S9(5) COMPUTATIONAL.
02 PCB4-NU-SENSESEG   PIC S9(5) COMPUTATIONAL.
02 PCB4-KEY-FB        PIC X(??) .

```

Figure 176 Sample COBOL Program Linkage Section to Access IMS/DB or DL/I Databases (Page 2 of 2)

The called program must also code the following in the Procedure Division to resolve the address of the list of associated PCBs to the PCB masks:

```

PROCEDURE DIVISION USING PCBLIST-PTR.
    SET ADDRESS OF PCB-PTRS TO PCBLIST-PTR.
    SET ADDRESS OF DB-PCB1 TO PCB1-PTR.
    SET ADDRESS OF DB-PCB2 TO PCB2-PTR.
    SET ADDRESS OF DB-PCB3 TO PCB3-PTR.
    SET ADDRESS OF DB-PCB4 TO PCB4-PTR.

```

Figure 177 Sample COBOL Program Procedure Division to Access IMS/DB or DL/I Databases

The COBOL program can now use the PCB masks (DB-PCB1, DB-PCB2, DB-PCB3, and DB-PCB4) to directly access the IMS/DB or DL/I databases.

COBOL Subroutine Considerations

When a COBOL subroutine is called, its LINKAGE SECTION must describe data areas that correspond to those passed in the parameter list. These are the data fields the COBOL external subroutine is USING upon entry.

The following is a z/OS VISION:Results program invoking an ANSI COBOL subroutine to move an input record to two WORKAREAs defined in the program.

```

FILE SYSIN CARDS    <---- z/OS
  RECORDAREA  80
WORKAREA
  REC_WORK_1  80
  REC_WORK_2  80
CALL COBPGM USING RECORDAREA REC_WORK_1 REC_WORK_2
PRINT REC_WORK_1
FIN

```

The following is a VSE VISION:Results program invoking an ANSI COBOL subroutine to move an input record to two WORKAREAs defined in the program.

```

FILE SYSIPT CARDS    <---- VSE
  RECORDAREA  80
WORKAREA
  REC_WORK_1  80
  REC_WORK_2  80
CALL COBPGM 8K USING RECORDAREA REC_WORK_1 REC_WORK_2
PRINT REC_WORK_1
FIN

```

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. COBTST.  
ENVIRONMENT DIVISION.  
.  
.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
LINKAGE SECTION.  
77 AREA1      PICTURE X(80).  
77 AREA2      PICTURE X(80).  
PROCEDURE DIVISION.  
COBENT.  
    ENTER LINKAGE.  
    ENTRY 'COBENT' USING AREA1 AREA2.  
    ENTER COBOL.  
PROCESS-REC.  
    MOVE AREA1 TO AREA2.  
    MOVE AREA2 TO AREA1.  
RETURN-LINKAGE.  
GOBACK.
```

Figure 178 F COBOL Example

ANSI COBOL also allows the entry statement.

If VISION:Results is to use this entry, COBENT (which is identified by the ENTRY 'COBENT' in the previous program) must be included in the link edit control statements.

```
//LKED.SYSIN DD *  
ENTRY COBENT  
NAME COBPGM
```

Figure 179 COBENT z/OS Example

```
COMPILE AND LINK  
// JOB  
// OPTION CATAL  
// PHASE COBPGM,*  
// EXEC FCOBOL  
.  
Cobol program  
/*  
    ENTRY COBENT  
// EXEC LNKEDT  
/&  
  
LINK (OBJECT MODULE OF COBOL PROGRAM  
    IN RELOCATABLE LIBRARY)  
// JOB  
// OPTION CATAL  
// PHASE COBPGM,*  
// INCLUDE object module name  
// ENTRY COBENT  
// EXEC LNKEDT  
/&
```

Figure 180 COBENT VSE Example

```

IDENTIFICATION DIVISION.
PROGRAM-ID. COBTST.
ENVIRONMENT DIVISION.
.
.
DATA DIVISION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
77 AREA1      PICTURE X(80).
77 AREA2      PICTURE X(80).
77 AREA3      PICTURE X(80).
PROCEDURE DIVISION USING AREA1 AREA2.
PROCESS-REC.
    MOVE AREA1 TO AREA2.
    MOVE AREA2 TO AREA3.
RETURN-LINKAGE.
GOBACK.

```

Figure 181 ANSI COBOL Example

If the PROCEDURE DIVISION USING format is used in an ANSI COBOL program, the ENTRY statement need not be included in the link edit JCL. However, you should always supply the ENTRY statement when the PROCEDURE DIVISION USING statement is the PROGRAM-ID name.

```

//LKED.SYSIN DD * <---- z/OS
ENTRY COBTST
NAME COBPGM

```

Note: If the LE option is chosen, the rest of this section does not apply.

Under VSE, ANSI COBOL, and VS COBOL II for VSE/ESA, programs must know whether they are main programs or called subroutines. These COBOL programs consider themselves to be main programs unless informed otherwise. The AUTOLINK feature of the linkage editor automatically includes the relocatable module, ILBDMNS0, that identifies the COBOL program as a main program. The module following must therefore be assembled as an object deck identifying the COBOL program as a called subroutine. This CSECT, when included at link edit time, defines the COBOL program as a called subroutine. A program called by VISION:Results is considered a subroutine.

```

// EXEC ASSEMBLY
ILBDMNS0 CSECT
DC X'FF000000000000000000000000000000'
END
/*

```

Note: This module can be cataloged in your object or relocatable library using a name other than ILBDMNS0, such as DUMYMNS0.

This version of ILBDMNS0 must be cataloged in the same phase with your COBOL program in the Core Image Library.

```

// JOB <---- VSE
// OPTION CATAL
PHASE COBPGM,*
INCLUDE COBTST object module
INCLUDE DUMYMNS0
ENTRY COBTST
// EXEC LNKEDT
/*
/&

```

The reference to ILBDMNS0 from the COBOL program is resolved by the AUTOLINK feature automatically unless the above is performed.

Under z/OS, VS COBOL II subroutines can be called by VISION:Results after the COBOL environment is set up. VS COBOL II and LE/370 COBOL require a COBOL run-time environment to be established prior to the first CALL of the COBOL program if the program is to be treated as a subroutine.

Failing to establish the environment can result in U1044, U1010, S0C1, or S0C4 abends, as well as reinitialization of the environment on every CALL (causing excessive CPU usage, and loss of data being maintained in COBOL working storage). The COBOL run-time environment is automatically established by VISION:Results if the DYLINSTL macro parameter COBENV=Y was specified. For more details, see the *Advantage VISION:Results for z/OS Installation Guide*.

If you do not want the automatic establishment of the COBOL environment, the VISION:Results program must call IBM routine ILBOSTP0 to set the environment prior to calling any VS COBOL II subroutine. The automatic establishment is the preferred method.

An additional DYLINSTL macro parameter, COB2NR=Y, is available to allow proper execution when calling COBOL II subroutines compiled with the NORES option. Because the COBOL run-time library is not used in a NORES application, specifying COB2NR overrides the COBENV automatic environment option. Specifying COB2NR on the OPTION statement overrides a DYLINSTL-specified COBENV for a given program.

COBOL Problem Areas

- An 0C1 occurs in a z/OS environment (or an 0C4 with IBM/VS 2.4 COBOL Compiler) if the COBOL program (external) has been compiled with the ENDJOB option; the input/output block (IOB) appears to have been released by the VS system. To eliminate this problem, recompile the subroutine without the ENDJOB option.
- An unpredictable system interrupt occurs at execution time if the VSE ANSI user does not make the subroutine a called program.
- See [COBOL Subroutine Considerations on page 281](#).

Assembler Subroutine Considerations

Upon entry to an Assembler external subroutine, the register contents are as follows.

Register	Contents
1	Address of the parameter list. The list is composed of fullword addresses of each data item passed in the VISION:Results parameter list. The last address in the list has its high order bit set to 1.
2-12	VISION:Results registers. They must be saved and restored by the external subroutine.
13	Address of an 18-fullword save area in the calling program. This can be used by the external subroutine to hold the VISION:Results registers.
14	Return address. The external subroutine returns to the calling program using register 14.
15	Entry point of the called external subroutine.

In addition to procedure logic functions, your external subroutines written in Assembler can read and write records. They can also CALL, LOAD, or LINK (z/OS) to other external subroutines. Under VSE, if your module has to LOAD any other phase, byte 36 (hex 24) of the VSE Partition Communications Region contains the address of the uppermost byte of memory used by VISION:Results.

Because SPIE (z/OS) and STXIT/PC (VSE) macros are used by VISION:Results for error recovery and analysis, avoid using these macros in an external subroutine.

The following example demonstrates a VISION:Results program invoking an Assembler external subroutine. VISION:Results calls a table search subroutine to locate a designated entry (ARGUMENT) in a table (TABLE) and places it in the work area ENTRY. The ARGUMENT, TABLE, and ENTRY data area addresses are being passed to one subroutine, TBLSRCH.

```
FILE ARMSTR FB 520 2600
.
.
FILE SYSIN CARDS          <---- z/OS
.
.
FILE SYSIPT CARDS         <---- VSE
.
.
WORKAREA
TABLESP 2000
ARGUMENT 10
WORKAREA
ENTRYSP 40
.
.
```

Figure 182 Assembler External Subroutine

(load table information into work area TABLESP)

```
.
.
.
CALL TBLSRCH USING ARGUMENT TABLESP ENTRYSP
.
.
.
```

```
TBLSRCH CSECT
      STM 14,12,12(13) *SAVE REGISTERS
      LR 3,15          *ENTRY POINT ADDRESS TO BASE REG
      USING TBLSRCH,3  *ESTABLISH ADDRESSABILITY
      ST 13,SAVE+4     *SAVE CALLING PROGRAMS REG 13
      LA 5,SAVE        *ADDRESS OF CALLED SAVE AREA
      ST 5,8(,13)      *CALLED PGM SAVE AREA SAVED IN
*
      LR 13,5          *CALLING PROGRAM SAVE AREA
      LM 9,11,0(1)     *REG 13 POINT TO CALLED SAVE AREA
*                      *LOAD PARAMETER ADDRESSES
*                      REGISTER 9 = ADDRESS OF ARGUMENT
*                      REGISTER 10 = ADDRESS OF TABLE
*                      REGISTER 11 = ADDRESS OF ENTRY
      .
      .
      .
      (table search logic)
      .
TBLEXIT L 13,SAVE+4    *RESTORE CALLER'S REGISTERS
      LM 14,12,12(13)
      BR 14           *RETURN TO CALLING PROGRAM
SAVE DC 18F'0'
      END
```

PL/I Subroutine Considerations

If a PL/I subroutine is called, you must call an Assembler language subroutine that supplies the necessary linkage to the PL/I program. An Assembler language subroutine called PL1EXIT has been supplied with VISION:Results in the source library created during the installation process.

PLIEXIT must be assembled and the object module placed in an object library.

z/OS PL/I Subroutine

```
//HLASM EXEC PGM=ASMA90,REGION=0M,TIME=(,40), X
// PARM=(OBJECT,NODECK,'LINECOUNT(48)', 'USING(MAP,WARN(3))', X
// TERM,'XREF(FULL)')
//SYSPRINT DD SYSOUT=*
//SYSPUNCH DD DUMMY
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSLIN DD DSN=OBJECT.LIB(PLIEXIT),DISP=SHR
//SYSTERM DD SYSOUT=*
//SYSIN DD DSN='YOUR.RESULTS.SOURCE.LIB(PLIEXIT)',DISP=SHR
```

Figure 183 z/OS PL/I Subroutine

VSE PL/I Subroutine

```
// DLBL IJSYSPH,'==OBJECT',0
// EXTENT SYSPCH,VVVVVV,1,0,1,20
// ASSGN SYSPCH,DISK,VOL=VVVVVV,SHR
// OPTION DECK
// EXEC ASSEMBLY
// PUNCH 'ACCESS SUBLIB=DYLLIB.OBJ'
// PUNCH 'CATALOG PLIEXIT.OBJ REPLACE=YES'
// (PLIEXIT source)
/*
// CLOSE SYSPCH,PUNCH
// DLBL DYLLIB,'DYL.LIBRARY'
// EXTENT SYS006,VVVVVV
// ASSGN SYS006,DISK,VOL=VVVVVV,SHR
// DLBL IJSYSIN,'==OBJECT'
// EXTENT SYSIPT,VVVVVV
// ASSGN SYSIPT,DISK,VOL=VVVVVV,SHR
// EXEC LIBR
/*
// CLOSE SYSIPT,READER
```

Figure 184 VSE PL/I Subroutine

The PL/I program can be compiled and link edited with the PLIEXIT program as illustrated in the following example.

z/OS PL/I Program with PLIEXIT

```
//COMP EXEC PGM=IEL1AA,REGION=0M,
// PARM='A(F),M,MAP,NC(E),NEST,NUM,
// OF,STG,X(F),NIS,LIST,OPT(2)'
//SYSPRINT DD SYSOUT=*
//SYSIN DD DSN=SOURCE.LIB(PLILKUP),DISP=SHR
//SYSLIN DD DSN=&&SYSLIN,DISP=(MOD,PASS),UNIT=SYSDA,
// SPACE=(CYL,(2,1,0)),DCB=(BLKSIZE=400)
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(2,1),,CONTIG),
// DCB=BLKSIZE=1024
//LKED EXEC PGM=IEWL,PARM='MAP,LET,LIST'
//SYSLMOD DD DSN=LOADLIB,DISP=SHR
//SYSLIB DD DSN=CEE.SCEELKED,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSPRINT DD SYSOUT=*
//OBJLIB DD DSN=OBJECT.LIB,DISP=SHR
//PLIMOD DD DSN=&&SYSLIN,DISP=(OLD,DELETE)
//SYSLIN DD *
INCLUDE OBJLIB(PLIEXIT)
INCLUDE PLIMOD
NAME PLILKUP
/*
```

Figure 185 z/OS PL/I Program Compiled and Link Edited with PLIEXIT

VSE PL/I Program with PLIEXIT

```
// OPTION CATAL
// PHASE PLILKUP,*
// INCLUDE PLIEXIT
// EXEC PLIOPT,SIZE=64K
// (PL/I source)
/*
// DLBL DYLLIB,'DYL.LIBRARY'
// EXTENT SYS006,VVVVVV
// ASSGN SYS006,DISK,VOL=VVVVVV,SHR
// LIBDEF OBJ,SEARCH=DYLLIB.OBJ
// LIBDEF PHASE,CATALOG=DYLLIB.PHASE
// EXEC LNKEDT
/*
```

Figure 186 VSE PL/I Program Compiled and Link Edited with PLIEXIT

The following example demonstrates a VISION:Results program invoking a PL/I external subroutine. The program calls a code lookup subroutine (PLILKUP) to locate a code (WKCODE) in a table and place the result in the work area (WKDESC).

```
FILE MASTER FB 520 2600
:
WORKAREA
WKCODE 4
WKDESC 32
:
:
(load code information into work area WKCODE)
:
:
CALL PLILKUP USING WKCODE WKDESC
:
:
FIN
```

Figure 187 Starting a PL/I External Subroutine

The following is an example of the PL/I program PLILKUP:

```

PLILKUP : PROC(PARM1) REORDER OPTIONS(MAIN);
DCL PARM1 PTR;
DCL 1 PARMLST BASED(PARM1),
    2 CODEPTR PTR INIT(NULL),
    2 DESCPTR PTR INIT(NULL);
DCL CODE1 BASED(CODEPTR) CHAR(4);
DCL DESC1 BASED(DESCPTR) CHAR(32);
DCL I FIXED BIN(15,0);
DCL 1 LIST (5) STATIC,
    2 CODE CHAR(4) INIT ('0011','0022','1133',
                        '2244','4567'),
    2 DESC CHAR(32) INIT ('POST OFFICE','SCHOOL','BANK',
                        'HOSPITAL','POLICE STATION');
DO I = 1 TO 5 UNTIL(LIST.CODE(I) = CODE1);
END; /* DO I = 1 ... */
IF I <= 5 THEN DESC1 = LIST.DESC(I);
    ELSE DESC1 = '';
END; /* PLILKUP */

```

Figure 188 Example of the PL/I Program PLILKUP

PLIEXIT Source Program

```

*****
***
***      PROGRAM: PLIEXIT
***      DATE WRITTEN: AUGUST, 1995
***      AUTHOR: COMPUTER ASSOCIATES
***
***      FUNCTION: THIS PROGRAM IS CALLED FROM A VISION:RESULTS
***                  PROGRAM WHICH IN TURN CALLS A PL/I MAIN PROGRAM
***                  .
***                  ITS PURPOSE IS TO ELIMINATE THE NEED OF THE
***                  VISION:RESULTS PROGRAMMER TO SET UP ALL OF THE
***                  PARAMETERS NEEDED BY PL/I.
***
*****
PLIEXIT  START 0
          B      12(R15)
          DC      CL8'PLIEXIT '      PROG ID
          STM     R14,R12,12(R13)      SAVE CALLING PROGS REGISTERS
          LR      R11,R15              OUR BASE
          USING   PLIEXIT,R11
          ST      R13,SAVE+4           SAVE CALLING PROGS R13
          LA      R2,SAVE              ADDR OF OUR SAVE AREA
          ST      R2,8(R13)            IN CALLING PROGS AREA
          LR      R13,R2
          ST      R1,PARM280           ADDR OF PARM LIST FROM VISION:RESULTS
          LA      R1,PARM280           ADDR OF ADDR OF PARM LIST
          ST      R1,PARMPLI
          LA      R1,PARMPLI           ADDR FOR PL/I
          ST      R1,ALIST
          LA      R1,ALIST             ARGUMENT ADDR
          L       R15,=V(PLICALLB)    CALL PL/I LOAD MODULE
          BALR    R14,R15
          L       R13,SAVE+4           RESTORE CALLING PROGS REGISTERS
          LM      R14,R12,12(R13)
          BR      R14                  RETURN TO CALLING PROGRAM
SAVE      DS     9D                   REGISTER SAVE AREA
          LTORG
PARM280   DS     A
PARMPLI   DS     A
ALIST     DC     A(*)                  ADDR OF ARGUMENT LIST
          DC     A(ISALEN)             ADDR OF LENGTH OF ISA STORGE
          DC     A(ISAADDR)            ADDR OF ISA POINTER
          DC     A(0)                  TASK ISA - NOT USED
          DC     A(0)                  NUMBER OF CONCURRENT SUBTASKS - NONE
          DC     A(OPTIONS)            ADDR OF OPTIONS WORD
          DC     A(HPSIZE)             ADDR SIZE OF HEAP

```

Figure 189 PLIEXIT Source Program (Page 1 of 2)

```

                DC      A(HEAP)                ADDR OF HEAP
                DC      A(HEAPINC)             ADDR OF HEAP INCREMENT
                DC      A(0)                   ADDR OF SUBTASK HEAP INCREMENT
                DC      A(ISAINC)              ADDR OF ISA INCREMENT
                ORG      *-4
                DC      X'80'                  END OF LIST
                ORG
ISALEN          DC      A(1024*14)             ISA LENGTH = 14K
ISAADDR          DC      A(ISASTOR)            ADDR OF ISA
                DS      0D
ISASTOR          DC      1024XL14'00'         ISA, 14K
HEAP             DC      1024F'0'             HEAP 4K
HPSIZE           DC      A(*-HEAP)            HEAP SIZE = 4K
HEAPINC          DC      F'4096'              HEAP INCREMENT 4K
ISAINC           DC      F'4096'              ISA INCREMENT 4K
*
OPTIONS          DC      AL1(NOREPORT+STAE,FREEHEAP+BELHEAP,0,0)
*
*              DEFINITIONS OF BITS IN OPTIONS BYTE
*
REPORT           EQU     X'80'                FIRST BYTE
NOREPORT          EQU     X'40'
SPIE              EQU     X'20'
NOSPIE            EQU     X'10'
STAE              EQU     X'08'
NOSTAE            EQU     X'04'
COUNT           EQU     X'02'
NOCOUNT           EQU     X'01'
*
FLOW              EQU     X'80'                SECOND BYTE
NOFLOW            EQU     X'40'
KEEPHEAP          EQU     X'20'
FREEHEAP          EQU     X'10'
ANYHEAP           EQU     X'08'
BELHEAP           EQU     X'04'
MKEPHEAP          EQU     X'02'
MPREHEAP          EQU     X'01'
*
MANYHEAP          EQU     X'80'                THIRD BYTE
MBELHEAP          EQU     X'40'
*
R1                EQU     1
R2                EQU     2
R11               EQU     11
R12               EQU     12
R13               EQU     13
R14               EQU     14
R15               EQU     15
END               PLIEXIT

```

Figure 189 PLIEXIT Source Program (Page 2 of 2)

Writing a Modify Module

A module written to use the MODIFY keyword of the FILE statements can be written in COBOL or Assembler. It must follow the suggestions and limitations for modules named in CALL statements described in this chapter. Control is passed to the module immediately following an input operation or immediately prior to an output operation. Upon entry to the user module, register 13 points to a save area and register 1 points to a parameter list.

To implement a modify module for a file, use the following phrase in the FILE statement:

```
[MODIFY modulename [nK] [SIZE nnnnn] [PARM (IOAREA | WORK |
!dataname!)]]
```

PARM

If you have supplied a PARM keyword in the FILE statement, the parameters are passed to the MODIFY subroutine. If no PARM is specified, a default parameter list is created based on the presence of a SIZE parameter.

SIZE

If there is no SIZE 2 specified, the default parameter list for MODIFY contains one entry, the address of the record. For an Assembler subroutine, you must reference the data using the address passed in the parameter list. For COBOL modules, you must define the record in the LINKAGE SECTION and specify this description in the ENTRY statement or in the PROCEDURE DIVISION USING statement. In either language, the user module must modify the record in place.

If a SIZE parameter is specified, the default parameter list for MODIFY contains two entries: the address of the record and the address of the work area. For an Assembler subroutine, you can set up addressability to these two areas by using the addresses in the parameter list.

For COBOL modules, you must describe the record and the work area in the LINKAGE SECTION and specify these descriptions in the ENTRY statement or in the PROCEDURE DIVISION USING statement.

For input files, the user module must act upon the data in the first parameter and place it in the second parameter. For output files, the data must be moved from the second parameter to the first. VISION:Results read/writes using the first area, but for logic purposes, is only able to reference the second area.

If you have coded your own PARM, the rules differ. However, if SIZE is specified, it is still necessary to move to or from the work area in the user module.

Writing an EXIT Module

A module to perform FILE EXIT processing can be written in either COBOL or Assembler. It must follow the suggestions and limitations outlined for modules named in CALL statements described in this chapter. The module is invoked whenever an I/O operation on the specified file is required. When the module is entered, register 13 points to a save area and register 1 points to a parameter list.

To implement an EXIT module for a file, use the following phrase in the FILE statement:

```
[EXIT modulename [nK] [SIZE nnnnn] [PARM (IOAREA | WORK |
!dataname!)]]
```

Figure 190 Writing an EXIT Module

PARM

If you have supplied a PARM keyword on your FILE statement, the parameter list corresponds to the PARM operands specified there. If there has been no parameter specified, then VISION:Results sets up a default parameter.

If you have not supplied a PARM keyword on your FILE statement, the parameter list contains three entries. The first entry is the area where the input record is to be placed or where the output record is to be written from. The second entry is a 1-byte status field allocated by VISION:Results. It is your responsibility to set this byte to E when an input file goes to the end of file. During the termination processing, VISION:Results places an E in this byte and invokes the routine one last time. It is your responsibility to allow for this and to make sure the file is closed. The third entry is a 17-byte area containing file characteristics as described in the FILE statement. The layout of this parameter is as follows.

Column	Description
1-8	file name
9-10	record format
11-12	logical record length in binary
13-14	block length in binary
15	operation type
	I input
	O output
	B both (IO was specified)
16	system number in binary
17	device type
	blank tape
	X'00' 2311
	X'01' 2314
	X'04' 3330
	X'08' 3340
	X'07' 3350
	X'08' 3375
	X'08' 3380

STATUS

VISION:Results always allocates a STATUS byte for every EXIT-processed FILE, even if you are providing your own PARM. You can specify a data name for this byte using the STATUS keyword in the FILE statement. For an input file, it is the user module's responsibility to place an E in this byte when the file goes to end of file. If this is not done, VISION:Results can go into a loop when an attempt is made to SORT this file.

When a user module is written, check the status byte on entrance to the module. If the byte contains an E, the module must close the file. This is because:

- During final wrap-up, VISION:Results moves an E to the status byte of each EXIT-processed FILE and then passes control to each EXIT module. This is done to give the user module an opportunity to close files during shutdown.
- Whenever a CLOSE command is executed, the user module is called with an E in the STATUS byte of the FILE statement.

Subroutines Link Edited as AMODE(31)

VISION:Results can be customized to support calling subroutines that have been linked with AMODE(31) and RMODE(any). Because the most recent releases of IBM COBOL can easily create such subroutines, VISION:Results allows you to run on a z/OS system, or VSE/ESA 1.3 or later, and CALL subroutines created with COBOL or Assembler that run above the 16-megabyte line.

Subroutines running above the 16-megabyte line can return to VISION:Results with either a BSM 0,14 or a BR 14 instruction. Subroutines link edited to run below the 16-megabyte line continue to work without modifications.

Chapter 14: Using Procedural Commands

This chapter shows you how to get in, around, and out of your VISION:Results program using the [GOTO](#), [NEXT](#), [PERFORM](#), [ACCEPT](#), [REJECT](#), [STOP](#), and [QUIT](#) commands.

Instructions in the procedure section of your VISION:Results program are typically executed one after the other:

```
MOVE FIELDA TO FIELDB  
COUNTER1=COUNTER1+1  
MOVE FIELDC TO FIELDDD  
.  
.  
.
```

First the MOVE statement is executed, then COUNTER1 is incremented by 1, followed by another MOVE. When there are no more instructions to execute or a special logic section (ON END, ON CHANGE, ON FINAL) other than ON ONE is encountered, an automatic ACCEPT occurs and the logic is exited. The output phase of the VISION:Results cycle is then entered.

In many of your programs, you can branch backward or forward to execute statements in other parts of the procedure. Or, you can terminate the run yourself when you have finished processing. This chapter shows you how to accomplish these tasks.

This chapter also explains how to use the Dynamic Allocation feature. Dynamic Allocation allows input and output files to be allocated without providing the JCL to do so in a z/OS operating environment, primarily to support APPC/MVS.

Details for using Dynamic Allocation can be found in [Dynamic Allocation on page 317](#), which also contains other commands affected by Dynamic Allocation and an explanation of error processing.

[Commands on page 296](#) discusses directing program execution through and out of procedure logic. [Figure 191](#) demonstrates the methods. [Figure 192 on page 299](#) shows how to use the PERFORM statement.

Example

```
FILE ACCFLEI    FB 80 800 SYS011;          <---- VSE
  ACCNUM 5 CH (ACCOUNT'NUMBER)    FILLER 56
  ACCAMT1 5 PD 2 A (AMOUNT'1)
FILE ACCFLEO OUTPUT FB 80 800 SYS021 FROM ACCFLEI

WORKAREA
  OVEFLAG 1 (OVER'200.00) VALUE ' ' REINIT
  STACODE 1 (ACCOUNT'STATUS) VALUE ' ' REINIT

CONTROL ACCNUM

PROCEDURE:
  IF ACCNUM EQ '99999' STOP ENDIF
  IF ACCNUM EQ '03001' THRU '03999' NEXT
    ELSE GOTO DELCHECK ENDIF
  CALL GETSTAT USING ACCNUM STACODE
  IF ACCAMT1 GT 200.00
    MOVE '**' TO OVEFLAG
  ENDIF
LISTDET:
  LIST ACCNUM ACCAMT1 OVEFLAG STACODE
  ACCEPT
DELCHECK:
  IF ACCNUM EQ '01029' REJECT ENDIF
*
ON CHANGE IN ACCNUM
  LIST ACCNUM SUM ACCAMT1 WITH 2 BEFORE AND 3 AFTER
ON FINAL
  LIST SUM ACCAMT1 'GRAND TOTAL'
```

Figure 191 Directing Program Execution Example

In this example, a file assumed to be in account number sequence is being processed. Records containing account numbers 03001 through 03999 are listed in the report. Any record going to the report with an amount over \$200.00 is flagged.

Additionally, the status of each selected account is being checked by exiting to a user routine that returns the status code of the account number being passed to it. This status code is also listed in the report. An output file of all records except those containing the account numbers 01029 and 99999 is written. When the 99999 record is encountered, the run terminates.

Commands

This section describes the commands.

GOTO

The command GOTO followed by a tagname causes execution to jump to the statement immediately following the tagname specified. You can branch backward or forward in your logic. Do not try to branch to logic that is part of ON CHANGE or ON FINAL logic or to branch to any logic that is coded after these kinds of logic.

In the example, the GOTO DELCHECK is branched to if the account number is not in the range 03001 through 03999. IF ACCNUM EQ 03001 THRU 03999 NEXT ELSE GOTO DELCHECK ...

A GOTO does not have to be part of an IF statement. It can be an independent statement:

```
MOVE FIELDA TO FIELDB
GOTO PROCB
```

NEXT

The keyword NEXT can appear only in an IF statement. It directs the execution for either a true or a false condition, but it is not required. It means that if the condition is met, the statement following the ENDIF terminator is to be executed. NEXT is used for readability; if left out, it is implied.

In the example, the statement specifies that if the account number is in the range of 03001 through 03999, the CALL GETSTAT statement is executed.

```
IF ACCNUM EQ 03001 THRU 03999 NEXT ELSE GOTO DELCHECK ENDIF
CALL GETSTAT USING ACCNUM STACODE
```

ACCEPT

The command ACCEPT specifies that the procedure logic is to be exited at that point and that the output phase of the VISION:Results cycle is to be entered. If a record or records are to be written or a record is printed because file print has been specified in the OPTION statement, these events take place.

In the example, an ACCEPT appears after the LIST statement identified by the tagname LISTDET. The program is exited at this time before falling through into the DELCHECK logic. No file print is in effect, but an output file (ACCFLEO) has been specified, so a record is now written to it.

Even if you do not code an ACCEPT in your program, an ACCEPT occurs under the following conditions:

- When the end of your logic statements has been reached:

```
.
.
.
MOVE FIELDA TO FIELDB
MOVE FIELDC TO FIELDDD
T1 'ACCOUNT REPORT'
```

- When special logic (not including ON ONE) such as ON CHANGE, ON FINAL, or ON END has been encountered.

In the prior example, during the DELCHECK logic, an implied ACCEPT exists after the IF ACCNUM statement. That is because the logic that follows is part of ON CHANGE IN ACCNUM. This implied ACCEPT causes all of the records going through this logic to be written to the output file (except 01029 records).

REJECT

The REJECT command specifies that the procedure logic is to be exited at that point and the output phase of the VISION:Results cycle is not to be entered. If the file print option is in effect, a record is not printed. Any files that are being written during the output phase are not written to in this cycle.

In the example, during the DELCHECK logic, the REJECT is used in an IF statement. If the account number is equal to 01029, the program is exited using REJECT. Any 01029 accounts are not written to the ACCFLEO file.

STOP

The STOP command terminates processing. When it is executed, the output phase of the VISION:Results cycle is bypassed. No record is printed if the file print option has been specified, and no records are written to any defined output files. If a report is being produced, the ON CHANGE IN and ON FINAL logic is executed. The end-of-run statistics are printed and all active files are closed. The run is then terminated.

In the example, the STOP command is issued when the 99999 trailer record is read on the input file. This record is not written to the ACCFLEO because the output phase is bypassed. The ON CHANGE IN ACCNUM logic is executed one last time to list the subtotals for the last detail series. Then the ON FINAL logic is executed to list the grand totals. The I/O counts are printed, files are closed, and the run is terminated.

STOPALL

The STOPALL command terminates an entire VISION:Results run. When it is executed, the output phase of the VISION:Results cycle is bypassed and the ON CHANGE IN and ON FINAL logic is executed (like a normal STOP operation); in addition, no further processing occurs on the existing request or any subsequent request that is part of this VISION:Results run.

QUIT

Like the STOP command, QUIT terminates further processing, but does so without executing any ON CHANGE IN or ON FINAL logic. This is especially useful when there are no more input records to process or when the program logic decides to bypass a report or an entire step. All active files are closed and the run is terminated.

QUITALL

Like the STOPALL command, QUITALL terminates an entire VISION:Results run, but does so without executing any ON CHANGE IN or ON FINAL logic (like a normal QUIT operation). In addition, no further processing occurs on any subsequent request that is part of this VISION:Results run.

PERFORM

The PERFORM statement executes an instruction or series of instructions located somewhere else in your program and then returns to the next statement after the PERFORM statement. This is particularly useful when you need to execute the same series of instructions from various points in your program.

The format of this statement is as follows:

```
PERFORM tagname1 [TO tagname2]
      [VARYING name FROM initial TO ending [BY increment]]
```

Figure 192 PERFORM Statement Example

where tagname1 is the tagname that identifies the first instruction to be performed, and tagname2 is the tagname that must immediately follow the last instruction to be executed before returning to the next statement after the PERFORM statement.

The VARYING keyword can be used as a counter for the PERFORM statement. Starting from an initial value, you can PERFORM this statement until an end value. The name entry must be a field name. The initial, ending, and increment values can be either field names or literals and can contain up to nine digits to the right of the assumed decimal point. The increment can be positive or negative. If the BY increment value is not specified, the default is 1.

```
FILE ACCFLEI FB 80 800
  ACCNUM 5 CH FILLER 56 ACCAMT1 5 PD 2
FILE SYSIN CARDS
  ACCNUMT 5 CH
FILE ACCFLEO OUTPUT FROM ACCFLEI FB 80 800

READMST:
  READ ACCFLEI
READMSTX:
READTRAN:
  READ SYSIPT
COMPAREMT:
  IF ACCNUM EQ ACCNUMT
    PERFORM CHGMST TO WRITEMSTX
    PERFORM READMST TO READMSTX
    GOTO COMPAREMT ENDIF
  IF ACCNUM LT ACCNUMT
```

Figure 193 VARYING Keyword in the PERFORM Statement Example (Page 1 of 2)

```

        PERFORM WRITEMST TO WRITEMSTX
        PERFORM READMST TO READMSTX
        GOTO COMPAREMT ENDIF
    GOTO READTRAN
CHGMST:
    ACCAMT1=ACCAMT1*1.10
WRITEMST:
    WRITE ACCFLEO
WRITEMSTX: ...

```

Figure 193 **VARYING** Keyword in the PERFORM Statement Example (Page 2 of 2)

In the example, a file in account number sequence is read and updated. The transaction file (SYSIN) contains the account numbers of those records that are to be modified. The modification consists of increasing the ACCAMT1 field by 10%. There can be more than one account with the same account number.

The PERFORM statement is employed in several places. Initially, it is coded to cause execution of the instructions that modify the master record and write it:

```
PERFORM CHGMST TO WRITEMSTX
```

Here, the instructions that follow the CHGMST tagname are executed until WRITEMSTX is encountered. Then, control returns to the statement following the PERFORM statement, which happens to be another PERFORM statement (PERFORM READMST TO READMSTX).

You can issue your PERFORM statement as part of an IF statement, or it can be an independent statement.

If tagname2 is not specified, PERFORM starts at tagname1 and continues to the next tag encountered.

You can increment a variable while the PERFORM is taking place. The variable is given an initial value before the loop is first performed. At the end of each cycle through the loop, the variable is incremented and compared against the ending value. As soon as it is greater than (or less than if the increment is negative) the ending value, the loop is terminated.

PERFORM Considerations

You can branch out while you are in the middle of a series of instructions using PERFORM, but you must eventually get yourself back to the TO tagname that was specified in the PERFORM statement.

- It is illegal at detail time to PERFORM logic that appears after ON CHANGE or ON FINAL.
- You can do a PERFORM within a PERFORM, but the second PERFORM must get to its exit point and return to the original PERFORM, which must also exit using the TO tagname.

READ filename

The READ command causes a record to be read from the specified file. VISION:Results normally reads all sequential files when it goes through the automatic cycle. If a READ command is specified for any file, that particular file is read only when the READ command is executed.

When the specified file goes to end of file, the status byte is set to E. Issuing READ commands for any or all files does not affect when VISION:Results terminates. When there is more than one file in the run, CA recommends that you check the status byte after every READ command.

When the READ command is used, the input buffer area is not initialized until the first READ command executes. Any attempt to move data to this area prior to the READ command being issued results in an 0C4 ABEND or otherwise unpredictable results.

For any input file, the record area cannot be referenced until a record has been read. With a VSAM file using any kind of random retrieval, the space is not allocated for a record being read until the record has been found. This means that any attempt to reference a record that a VSAM READ could not find leads to unpredictable consequences, probably an abend.

VISION:Results automatically reads all sequential files that do not have READ specified. However, it does not automatically read any IO, RANDOM, or SKIP files. To read any of these file types, you must code a READ command.

WRITE filename

The WRITE command causes a record to be written to the specified file. VISION:Results normally writes to all sequential files when it goes through the automatic cycle. If a WRITE command is specified for any file, that particular file is written to only when the WRITE command is executed.

VISION:Results automatically writes to all sequential files that do not have WRITE specified. VISION:Results does not automatically write to any IO files. To write to this file type, you must code a WRITE command.

CLOSE filename

CLOSE is valid only with EXIT-processed files. It causes an E to be moved to the STATUS data name byte associated with the specified file. Then it issues a CALL to the EXIT module, passing the specified parameter list (specified by PARM in the FILE statement or by default). It is not necessary to close EXIT-processed files. VISION:Results automatically calls the EXIT module at end of job to close the file. If you need to reprocess a file in the same program that created it, a CLOSE is required.

For z/OS, if you want to sequentially process many different files in the same program using the same DD name, use the following steps in conjunction with the CLOSE command:

1. Specify DYNAM on the FILE statement containing EXIT DYLIU.
2. Include FREE=CLOSE in the JCL which will be used to allocate the EXIT DYLIU file.
3. Issue the ALLOCATE command for the EXIT DYLIU file.
4. Process the file.
5. Issue the CLOSE command for the file.
6. Change the DSN in the JCL used to allocate the file, to identify the new file to process.
7. Re-execute steps 3 through 6 for each additional file to process.

Procedures

This section describes the procedures.

ON ONE, ENDONE

To execute a procedure only once in a program, preface your statements with ON ONE and terminate them with ENDONE. When execution reaches one of these sections for the first time, the statements within it are executed. The next and subsequent times through, they are bypassed and the statement following the ENDONE terminator is executed.

```
FILE SYSIN CARDS
  CARDATE 6 NU  CARACCT 5
FILE ACCFLEI INPUT FB 80 800
  ACCNUM 5 1 CH  FILLER 56  ACCAMT1 5 PD 2

WORKAREA
  DATHOLD 6 NU D
  ACCHOLD 5

ON ONE
  MOVE CARDATE TO DATHOLD
  MOVE CARACCT TO ACCHOLD
ENDONE
  IF ACCNUM EQ ACCHOLD NEXT ELSE REJECT ENDIF
LIST ACCNUM (ACCOUNT'NUMBER) ACCAMT1 (AMOUNT'1)
ON FINAL
  LIST SUM ACCAMT1 'ACCOUNT TOTAL' WITH 3 BEFORE
T1 'ACCOUNT REPORT' WITH 2 AFTER
T1+1 ACCHOLD
T1+124 DATHOLD
FIN
05308103004
```

Figure 194 ON ONE and ENDONE Example

In this z/OS example, the instream data is being read to obtain the date to go in the report heading and to obtain the account number to be selected and reported on. ON ONE logic moves the date and account number to a hold area the first time through the program logic. On the next cycle, the IF statement following the ENDONE terminator is the first instruction executed.

If you code a READ immediately for the SYSIN in the ON ONE section rather than have it read as part of the automatic cycle, a problem occurs because the instream data never reaches end of file as it does on the second program cycle under control of the automatic read. Because VISION:Results does not terminate a run automatically unless all input files are at end, your program goes into a loop when the account file is completed. To prevent this, put in additional coding to check and terminate the run when the account file is exhausted.

ON ONE Considerations

- If you branch out of an ON ONE procedure, the statement following the ENDONE is executed the next time through. You cannot execute any incomplete ON ONE statements.
- You cannot code a SORT statement within ON ONE logic.
- You can have a maximum of 100 ON ONE procedures within your program.

ON END OF INPUT

VISION:Results normally initiates termination procedures when all input files defined in the program reach end of file. It then executes all ON CHANGE IN and ON FINAL (report) logic if present in the program, closes all active files, prints run statistics, and terminates.

You can do additional processing before VISION:Results finishes everything. You can print a message, write additional records to an output file, or pass more data to the report. You can accomplish this by including ON END OF INPUT procedure logic in your program. If you do this instead of executing termination activities, VISION:Results enters your program at this point, allowing you to do any additional processing. You can print data, write data, report on data, branch to other logic, and so on.

During ON END OF INPUT logic, you can also exit your program using [ACCEPT](#), implied [ACCEPT](#), or [REJECT](#). The regular output phase of the VISION:Results cycle occurs, and the logic is entered at the ON END OF INPUT procedure logic so you can continue processing for as long as necessary. For a complete description of the automatic cycle, see *Advantage VISION:Results for z/OS Getting Started*.

It is important that you terminate the run using a STOP command when you are finished. VISION:Results has no other way of knowing when you want to terminate.

The following program shows the use of ON END OF INPUT logic:

```
FILE ACCFLEI INPUT FB 80 800 SYS011
  ACCRECI 80 1
  ACCNUM 5 1 CH FILLER 56 ACCAMT1 5 PD 2
  ACCAMT2 5 PD 2 ACCAMT3 5 PD 2
FILE ACCFLEO OUTPUT FROM ACCFLEI FB 80 4000 SYS021

WORKAREA
  AMT1HOLD 5 PD 2 VALUE 0
  AMT2HOLD 5 PD 2 VALUE 0
  AMT3HOLD 5 PD 2 VALUE 0
  ENDSW 1 VALUE ' '
  ATLIT 13 VALUE 'ACCOUNT TOTAL'
  FTLIT 13 VALUE 'FINAL TOTAL'

CONTROL ACCNUM
PROCEDURE:
  AMT1HOLD=AMT1HOLD+ACCAMT1
  AMT2HOLD=AMT2HOLD+ACCAMT2
  AMT3HOLD=AMT3HOLD+ACCAMT3
  LIST ACCNUM (ACCOUNT'NUMBER) ACCAMT1 (AMOUNT'1)
    ACCAMT2 (AMOUNT'2) ACCAMT3 (AMOUNT'3)
ON CHANGE IN ACCNUM
  LIST ACCNUM SUM ACCAMT1 SUM ACCAMT2 SUM ACCAMT3
  ATLIT ( )
ON FINAL
  LIST SUM ACCAMT1 SUM ACCAMT2 SUM ACCAMT3
  FTLIT AT ATLIT
FIN
```

Figure 195 ON END OF INPUT logic

VISION:Results reads a sequential input file (ACCFLEI) that is in account number sequence. Amounts are accumulated in a work area (AMT1HOLD, AMT2HOLD, AMT3HOLD), and record fields are listed in the report (LIST). After the detail line is listed, the ON CHANGE IN ACCNUM procedure is encountered. Because this logic is not executed as part of regular program logic, the program is now exited using an implied [ACCEPT](#).

The output phase of the cycle is entered where the input record is written to the output file (ACCFLEO) as specified in the FILE statement. After this, a new cycle is started where VISION:Results reads another input record and enters the program logic. At the end of the input file, VISION:Results executes the ON CHANGE IN ACCNUM for the last time and executes the ON FINAL logic to list the grand totals on the report. VISION:Results closes all of the opened files, prints run statistics, and terminates.

At this point, assume you want to write a trailer record to the output file before the run terminates. This record contains the account number of 99999 and the accumulated amount fields obtained from the work area. This is where you need the ON END OF INPUT logic:

```
.
.
ON END OF INPUT
  IF ENDSW EQ 'Y' STOP ENDIF
  MOVE 'Y' TO ENDSW
  MOVE SPACES TO ACCRECI
  MOVE '99999' TO ACCNUM
  MOVE AMT1HOLD TO ACCAMT1
  MOVE AMT2HOLD TO ACCAMT2
  MOVE AMT3HOLD TO ACCAMT3
  ACCEPT
ON CHANGE IN ACCNUM
.
.
```


The ON END OF INPUT logic is entered when end of file is detected on the input file. This logic builds the trailer record in the input area and then an [ACCEPT](#) occurs, causing the record to be written during the output phase of the cycle. The program is then entered again at the ON END OF INPUT procedure, where processing is now terminated using [STOP](#) because ENDSW contains a Y.

Logic preceding an ON END OF INPUT procedure does not fall through into this procedure. If you have not coded an [ACCEPT](#), [REJECT](#), or branch, an [ACCEPT](#) occurs. In the preceding example, LIST logic immediately precedes the ON END OF INPUT section but an [ACCEPT](#) occurs after the LIST.

ON END OF INPUT Considerations

ON END OF INPUT cannot be used in the following circumstances:

- If there is any random, skip, or sequential (VSAM IO only) mode processing in the program. Under these processing modes, an end of file condition never occurs.
- If the following keywords are used in the program:
 - SORT
 - EXIT
 - MODIFY
 - USE
 - VFILE
 - DUMMYThese keywords utilize dummy or EXIT files internally, so an end of file condition never occurs.
- If a request consists only of input FILE statements that contain the EXIT, PDS, SYSIN, or MODIFY keywords, or USE statements referring to a PICNSAVE idname from a prior request.
- If it is a MATCH request, ON END OF INPUT cannot be used.
- If the SUM, SUMn, or PRIOR of any field in ON END OF INPUT logic is referenced. This type of referencing should only be done during ON CHANGE and ON FINAL logic when the report writer is in control and report total areas can be accessed.

For information about ON END OF SORTING, see [Chapter 12: Using the SORT Command](#).

Indexing

VISION:Results provides four indexes. They are 2-byte binary fields called INW, INX, INY, and INZ, which are initialized to zero. If used, make sure the index contains a valid value. Additionally, you can create your own indexes using the OPTION statement USERINXnn dataname (for more information, see [User-Defined Index Fields on page 77](#)).

Use these indexes to do some table manipulation, loop through several related fields in a record, or manipulate fields in a variable location within a record. The format for indexes is:

```
dataname (index) or dataname(index)
```

Figure 196 Format for Indexes

VISION:Results uses the attributes of the specified data name (for example, size, type, decimals) and determines the actual memory location of the field to be used by adding the contents of the index to the location of the specified data name. The index must be enclosed within parentheses when used with a data name.

For example, when the value of the index (INW) is zero, the following statement refers to the field defined by TRAILER.

```
TRAILER (INW) or TRAILER(INW)
```

However, when index (INW) contains the value 10, the same statement refers to a location 10 bytes beyond the data name TRAILER but having the same size, type, and decimals.

It is your responsibility to set the value in the index. You can do this by moving a value into the index. For example,

```
MOVE 0 TO INX
```

Or, you can set the value in the index by doing arithmetic on the index:

```
INX=INX+10
```

You can use MOVE to set up the initial value for the index or to set the index when it is a constant. However, for table manipulation or looping through related fields in a record, you probably want to do arithmetic on the index to get the appropriate value.

You determine what value to put in the index based on your particular application. In most cases, the index value starts at zero and is increased by the length of each table entry or repeated fields.

For example, you can have a file that contains 12 months of data. To get the total of all 12 months, you can code a data name for each month and add the 12 of them together or you can use indexing. Suppose the monthly totals are all 5-byte packed fields with two decimals. Supply a data definition for the first month's accumulator plus a filler for the remaining 11 months:

```
MONTHS 60
REDEFINE AT MONTHS
MONTHTOT 5 PD 2
FILLER 55
```

You also need to define a work area to hold the year's total:

```
YEARTOT 6 PD 2 VALUE 0 REINIT
```

REINIT was specified because in this case you want the annual total to start at zero for each record.

The following code causes the 12 monthly totals to be accumulated:

```
.
.
MOVE 0 TO INX
LOOP:
YEARTOT=YEARTOT+MONTHTOT(INX)
INX=INX+5
IF INX LT 60
GOTO LOOP
ENDIF
CONTINUE:
.
.
```

When control reaches the tagname CONTINUE, YEARTOT contains the total of the 12 accumulators. On the first entry at tagname LOOP, INX contains 0 and the first execution of the arithmetic statement causes the field defined by MONTHTOT to be added to the accumulator. The value 5 was added to INX because this is the length of the monthly total area.

On the second entry at tagname LOOP, INX contains 5 and the execution of the arithmetic statement causes whatever is 5 bytes beyond MONTHTOT, or in this case, the second month's total, to be added to the year's total. This repeated addition occurs until the value of INX is 60 or greater. When the value of INX reaches 60, all 12 months have been added and it stops. The 12-month total is now in YEARTOT.

Because VISION:Results does not check to ensure the validity of the index being used, make sure the index value is correct. If the value in the index is negative, negative indexing results. It is possible to refer to a field prior to a defined field by using a negative index. If the value in the index becomes larger than 32767, VISION:Results treats it as a negative value.

Commands Used with Indexing

Use indexes on the following commands only:

IF

```
IF ADDRNAME (INX) EQ 'JONES' GOTO ABC ENDIF
IF FIELD A GE FIELD B (INW) GOTO FOUND ENDIF
IF FIELD A (INX) EQ FIELD B (INW) GOTO DONE ENDIF
```

Both operands can be indexed.

MOVE

```
MOVE ENTRY TO TABLE (INX)
MOVE ENTRY (INX) TO TABLE (INW)
MOVE 0 TO INW
MOVE ADDRNAME LENGTH 20 TO ARGUMENT (INX)
MOVE ENTRY (INX) LENGTH 6 TO TABLE (INX)
```

Both from and to operands, as well as the LENGTH operand for variable-length moves, can be indexed.

COMBINE

Both operands, as well as the INTO operand, can be indexed.

ARITHMETIC

```
INW=INW+20
INX=3
INY=INX+INW
BALANCE=TOTAL (INW) +12
TOTAL=AMOUNT (INX) +AMOUNTYR (INW)
TOTAL ( INY) =AMOUNT (INX) +AMOUNTYR (INW)
```

All operands involved in an arithmetic expression can be indexed.

PRINT

All data names in a PRINT statement can be indexed.

LIST

Any non-control break data name can be indexed.

Indexing Restrictions

When using indexes, you cannot use one of the SUM qualifiers with an indexed field. To do this, you must move the indexed field to a non-indexed WORKAREA dataname and use the WORKAREA dataname with SUM.

Listing (LIST) an indexed control break field causes a validation error.

TRANSLATE

Use the TRANSLATE command to:

- Replace all occurrences of one character with a different character.
- Convert from uppercase to lowercase or vice versa.
- Convert from ASCII to EBCDIC or vice versa.
- Replace a variety of different hex or character values with new values of your choice.

Data can be translated in place, or it can be translated into another area leaving the original data unchanged.

The TRANSLATE statement, in its simplest form, looks like this:

```
TRANSLATE dataname1 FROM literal1 TO literal2
```

Where:

dataname1 is any field not longer than 256-bytes.

literal1 is any valid character, numeric, or hex literal.

literal2 is any valid character, numeric, or hex literal having the same length as literal1.

Dataname1 FROM

Example 1

```
TRANSLATE DATA1 FROM ' ' TO 0
```

This statement causes all spaces in the field DATA1 to be changed to zeros.

Example 1 can also be coded this way:

```
{SPACE }
TRANSLATE DATA1 FROM {SPACES} TO { '0' }
                      {BLANK}  { 0 }
                      {BLANKS} {X'F0'}
```

The figurative constants SPACE, SPACES, BLANK, and BLANKS have an implied length of 1 when used in a TRANSLATE instruction. Therefore, the corresponding TO or FROM value must also be 1-byte long when these special constants are used.

If more than one value needs to be changed, all FROM values are coded together as one or more literals and the corresponding TO values are coded in the same fashion.

Example 2

```
TRANSLATE DATA2 FROM X'818283' TO 'ABC'
```

This statement causes a hex 81 to be changed to an A, a hex 82 to be changed to a B, and so on.

Example 3

```
TRANSLATE DATA3 FROM X'4020' 'ABC' TO 0 BLANK X'818283'
```

This statement causes a hex 40 to be changed to a zero, a hex 20 to be changed to a space, an A to be changed to a hex 81, and so on.

INTO dataname2

This statement causes the data in dataname1 to be translated into dataname2, leaving dataname1 unchanged.

Example

```
TRANSLATE DATA1 INTO DATA4 FROM ',' TO '.'
```

or

```
TRANSLATE DATA1 FROM '/' TO '-' INTO DATA4
```

Dataname2 must be the same length, type, and number of decimals (if applicable) as dataname1.

The INTO statement can be placed either following dataname1 or following literal2.

UPPER, LOWER

Data conversion between these two representations can be accomplished automatically by coding:

```
TRANSLATE dataname1 FROM UPPER TO LOWER
```

Figure 197 UPPER, LOWER Example 1

or

```
TRANSLATE dataname1 FROM LOWER TO UPPER
```

Figure 198 UPPER, LOWER Example 2

The INTO statement can also be used.

ASCII, EBCDIC

Data conversion between these two representations can be accomplished automatically by coding:

```
TRANSLATE dataname1 FROM ASCII TO EBCDIC
```

Figure 199 ASCII, EBCDIC Example 1

or

```
TRANSLATE dataname1 FROM EBCDIC TO ASCII
```

Figure 200 ASCII, EBCDIC Example 2

The INTO statement can also be used.

TRANSLATE dataname1 [INTO dataname2]

This form of TRANSLATE is used only after you have dynamically modified the contents of the standard translate table.

The label DYLTRAN addresses the first byte of the table. Indexing can be used to point to other specific locations. DYLTRAN has an implied length of 1, which can be overridden by using the LENGTH modifier.

Example

Reverse the digits in a 6-character field.

```
MOVE X'050403020100' TO NEWDATA
MOVE OLDDATA TO DYLTRAN LENGTH 6
TRANSLATE NEWDATA
```

If OLDDATA contains BCDE21, NEWDATA becomes 12EDCB.

EDIT

Use the EDIT command to insert special characters (such as . and /) in a number, or to replace leading zeros with some other character (such as a space), usually for the purpose of printing the value more legibly. A VISION:Results supplied pattern can be used to control the edit, or you can supply your own.

Numeric fields that are listed can be edited automatically, in a variety of ways, through the use of an edit code in the field definition (see *Advantage VISION:Results for z/OS Getting Started*). Therefore, use the EDIT command if:

- A field needs to be edited in some special, non-standard way.

- A standard edit is needed, but the edited field is to be used in an output file or a work area.

Non-Standard Edits

The EDIT command, in its simplest form, looks like this:

```
EDIT dataname1 INTO dataname2
```

Figure 201 The EDIT Command

Where:

dataname1 Must be a packed decimal field, 1 to 16 bytes long.

dataname2 Must be a character field that contains the edit mask. See the IBM manual *Principles of Operation* for specifics on creating an edit mask.

Either data name can be indexed.

The EDIT operation moves the digits in dataname1 into the dataname2 field according to the edit pattern. The pattern itself is destroyed.

To ensure that the edit mask is inserted in the receiving field (dataname2) each time before editing, code your edit as follows:

```
EDIT dataname1 INTO dataname2 USING hexliteral
```

In this case, hexliteral must be a hexadecimal literal defining an edit mask from 2 to 9 bytes long and dataname2 must be the same length as the mask:

```
ex. X'4020216B20202060'
```

The INTO clause can also follow the literal.

Example 1—Check Protection

```
EDIT NETPAY INTO CHKAMT  
      USING X'5C20202020214B2020'
```

If NETPAY is a 4-byte packed decimal field containing the value 0030975, the CHKAMT (which must be a 9-byte character field) contains ***309.75.

A value less than 100 (82, for example) prints as *****.82.

Example 2—Telephone Number Edit

```
WORKAREA  
  PHONEMASK 7 VALUE X'214D2020205D40'  
  FILLER 8 VALUE X'2020206020202020'  
  .  
  .  
  PRTELNO 15  
  .
```



```
MOVE PHONEMASK LENGTH 15 TO PRTELNO
EDIT PHONE INTO PRTELNO
```

If PHONE is a 6-byte packed decimal field containing the value 08005551212, then PRTELNO contains (800) 555-1212 right-aligned in the field.

Standard Edits

Edit codes can be used in the EDIT statement to produce a variety of edited result fields. The general form of this instruction is:

```
EDIT dataname1 INTO dataname2 USING [edit code]
```

Where:

- dataname1** Must be a numeric (1 to 16 bytes), packed (1 to 10 bytes), or binary (1 to 4 bytes) field.
- dataname2** Must be a character field at least the size of dataname1. For all edits except D and S, the low order (rightmost) byte of dataname2 is blank if the value is zero or more, and - if less than zero.

Restrictions

- D edit** Dataname1 must be a 4-byte packed or 6-byte numeric field with no decimals.
- S edit** Dataname1 must be a 5-byte packed or 9-byte numeric field with no decimals.

Edit Codes

The edit codes you can supply in your field definition to override the default edit format (P edit code) are as follows.

Edit Code	Description	Examples
E	Edit with zero suppression, commas, decimal insertion, and negative sign to the right of the field. Blank if zero.	001234.56 prints as 1,234.56 000000.00 prints as (blank)
Y	Leading '-'. For use in data name definitions, LIST, EDIT, and LTD (Letterwriter) statements. Edit with zero suppression, commas, decimal insertion, and negative sign to the left of the number. Blank if zero.	-001234.56 prints as -1,234.56 000000.00 prints as (blank)
NE	Leading '-'. Applies only to data name definitions and LIST and EDIT statement overrides. Edit with zero suppression, commas, decimal insertion, and negative sign to the left of the number. Blank if zero.	-001234.56 prints as -1,234.56 000000.00 prints as (blank)

Edit Code	Description	Examples
DE	Leading '\$'. Applies only to data name definitions and LIST and EDIT statement overrides. Edit with zero suppression, commas, decimal insertion, and negative sign to the right of the field. Float a currency sign (the default is \$) to the left of the most significant digit. Blank if zero.	-001234.56 prints as \$1,234.56- 000000.00 prints as (blank)
F	Leading '\$'. For use in data name definitions, LIST, EDIT, and LTD (Letterwriter) statements. Edit with zero suppression, commas, decimal insertion, and negative sign to the right of the field. Float a currency sign to the left of the field. Blank if zero.	-001234.56 prints as \$1,234.56- 000000.00 prints as (blank)
NDE	Leading '\$-'. Applies only to data name definitions and LIST and EDIT statement overrides. Edit with zero suppression, commas, decimal insertion, and negative sign to the left of the field. Float a currency sign to the direct left of the field. Blank if zero.	-001234.56 prints as -\$1,234.56 000000.00 prints as (blank)
G	Leading '\$-'. For use in data name definitions, LIST, EDIT, and LTD (Letterwriter) statements. Edit with zero suppression, commas, decimal insertion, and negative sign to the left of the field. Float a currency sign to the direct left of the field. Blank if zero.	-001234.56 prints as -\$1,234.56 000000.00 prints as (blank)
DNE	Leading '\$-'. Applies only to data name definitions and LIST and EDIT statement overrides. Edit with zero suppression, commas, decimal insertion, and negative sign to the direct left of the field. Float a currency sign to the left of the negative sign. Blank if zero.	-001234.56 prints as \$-1,234.56 00000000 prints as (blank)
K	Leading '\$-'. For use in data name definitions, LIST, EDIT, and LTD (Letterwriter) statements. Edit with zero suppression, commas, decimal insertion, and negative sign to the direct left of the field. Float a currency sign to the left of the negative sign. Blank if zero.	-001234.56 prints as \$-1,234.56 00000000 prints as (blank)
A	Edit with zero suppression, commas, decimal insertion, and negative sign to the right of the field. Print decimal point and zeros to the right of decimal. If field has no decimal positions and has a zero value, nothing prints.	001234.56 prints as 1,234.56 0000000000 prints as (blank)
X	Leading '-'. For use in data name definitions, LIST, EDIT, and LTD (Letterwriter) statements. Edit with zero suppression, commas, decimal insertion, and negative sign to the left of the field. Print decimal point and zeros to the right of decimal. If field has no decimal positions and has a zero value, nothing prints.	001234.56 prints as 1,234.56 -001234.56 prints as -1,234.56 0000000.00 prints as .00 0000000000 prints as (blank)
NA	Leading '-'. Applies only to data name definitions and LIST and EDIT statement overrides. Edit with zero suppression, commas, decimal insertion, and negative sign to the left of the field. Print decimal point and zeros to the right of decimal. If field has no decimal positions and has a zero value, nothing prints.	001234.56 prints as 1,234.56 -001234.56 prints as -1,234.56 0000000.00 prints as .00 0000000000 prints as (blank)

Edit Code	Description	Examples
DA	Leading '\$'. Applies only to data name definitions and LIST and EDIT statement overrides. Edit with zero suppression, commas, decimal insertion, and negative sign to the right of the field. Float a currency sign (the default is \$) to the left of the most significant digit. Blank if zero.	001234.56 prints as \$1,234.56 -001234.56 prints as \$-1,234.56- 0000000.00 prints as \$.00 0000000000 prints as (blank)
H	Leading '\$'. For use in data name definitions, LIST, EDIT, and LTD (Letterwriter) statements. Edit with zero suppression, commas, decimal insertion, and negative sign to the right of the field. Float a currency sign (the default is \$) to the left of the most significant digit. Blank if zero.	001234.56 prints as \$1,234.56 -001234.56 prints as \$-1,234.56- 0000000.00 prints as \$.00 0000000000 prints as (blank)
NDA	Leading '\$-'. Applies only to data name definitions and LIST and EDIT statement overrides. Edit with zero suppression, commas, decimal insertion, and negative sign to the left of the field. Float a currency sign to the direct left of the field. Blank if zero.	001234.56 prints as \$1,234.56 -001234.56 prints as \$-1,234.56 0000000.00 prints as \$.00 0000000000 prints as (blank)
J	Leading '\$-'. For use in data name definitions, LIST, EDIT, and LTD (Letterwriter) statements. Edit with zero suppression, commas, decimal insertion, and negative sign to the left of the field. Float a currency sign to the direct left of the field. Blank if zero.	001234.56 prints as \$1,234.56 -001234.56 prints as \$-1,234.56 0000000.00 prints as \$.00 0000000000 prints as (blank)
DNA	Leading '\$-'. Applies only to data name definitions and LIST and EDIT statement overrides. Edit with zero suppression, commas, decimal insertion, and negative sign to the direct left of the field. Float a currency sign to the left of the negative sign. Blank if zero.	001234.56 prints as \$1,234.56 -001234.56 prints as \$-1,234.56 0000.00 prints as \$.00 0000000000 prints as (blank)
L	Leading '\$-'. For use in data name definitions, LIST, EDIT, and LTD (Letterwriter) statements. Edit with zero suppression, commas, decimal insertion, and negative sign to the direct left of the field. Float a currency sign to the left of the negative sign. Blank if zero.	001234.56 prints as \$1,234.56 -001234.56 prints as \$-1,234.56 0000.00 prints as \$.00 0000000000 prints as (blank)
Z	Edit with zero suppression, decimal insertion, and negative sign to the right of the field, but no commas. Blank if zero.	001234.56 prints as 1234.56 00000000 prints as (blank)
U	Leading '-'. For use in data name definitions, LIST, EDIT, and LTD (Letterwriter) statements. Edit with zero suppression, decimal insertion, and negative sign to the left of the field, but no commas. Blank if zero.	00123456 prints as 1234.56 -00123456 prints as -1234.56 00000000 prints as (blank)
NZ	Leading '-'. Applies only to data name definitions and LIST and EDIT statement overrides. Edit with zero suppression, decimal insertion, and negative sign to the left of the field, but no commas. Blank if zero.	001234.56 prints as 1234.56 -001234.56 prints as -1234.56 00000000 prints as (blank)

Edit Code	Description	Examples
B	Edit with zero suppression, decimal insertion, and negative sign to the right of the field, but no commas. Blank if zero. If zero value, print decimal point and zeros to the right of decimal. If field has no decimal positions and has a zero value, nothing prints.	001234.56 prints as 1234.56 000000.00 prints as .00 000000000 prints as (blank)
Q	Leading '-'. For use in data name definitions, LIST, EDIT, and LTD (Letterwriter) statements. Edit with zero suppression, decimal insertion, and negative sign to the left of the field, but no commas. Blank if zero. If zero value, print decimal point and zeros to the right of decimal. If field has no decimal positions and has a zero value, nothing prints.	001234.56 prints as 1234.56 -001234.56 prints as -1234.56 000000.00 prints as .00 000000000 prints as (blank)
NB	Leading '-'. Applies only to data name definitions and LIST and EDIT statement overrides. Edit with zero suppression, decimal insertion, and negative sign to the left of the field, but no commas. Blank if zero. If zero value, print decimal point and zeros to the right of decimal. If field has no decimal positions and has a zero value, nothing prints.	001234.56 prints as 1234.56 -001234.56 prints as -1234.56 000000.00 prints as .00 000000000 prints as (blank)
P	Default edit code. Edit with decimal insertion, and negative sign to the right of the field, but no commas. If the field has no decimal positions and has a zero value, zeros print. If it has a zero value, zeros print.	001234.56 prints as 001234.56 000000000 prints as 000000000
W	Leading '-'. For use in data name definitions, LIST, EDIT, and LTD (Letterwriter) statements. Edit with decimal insertion, and negative sign to the left of the field, but no commas. If the field has no decimal positions and has a zero value, zeros print. If it has a zero value, zeros print.	-001234.56 prints as -001234.56 0000000.00 prints as 0000000.00 000000000 prints as 000000000
NP	Leading '-'. Applies only to data name definitions and LIST and EDIT statement overrides. Edit with decimal insertion and negative sign to the left of the field, but no commas. If the field has no decimal positions and has a zero value, zeros print. If it has a zero value, zeros print.	-001234.56 prints as -001234.56 0000000.00 prints as 0000000.00 000000000 prints as 000000000
D	Date edit of a 6-byte zoned decimal or 4-byte packed field.	120100 prints as 12/01/00
V	For use in data name definitions, LIST, EDIT, and LTD (Letterwriter) statements. Edit a date field containing a 4-digit year. The date field must be either an 8-byte zoned decimal or 5-byte packed field.	12012000 prints as 12/01/2000
D4	Applies only to data name definitions and LIST and EDIT statement overrides. Edit a date field containing a 4-digit year. The date field must be either an 8-byte zoned decimal or 5-byte packed field.	12012000 prints as 12/01/2000
S	Edit a social security number from a 9-byte numeric or 5-byte packed decimal field.	999999999 prints as 999-99-9999

Dynamic Allocation

Note: This section only applies to z/OS.

Dynamic Allocation is a z/OS operating system service that allows input and output files to be allocated without having to provide the JCL to do so. Rather, the executing program, with the aid of Dynamic Allocation (referred to by z/OS as SVC 99), allocates the required files.

Normally, to use dynamic allocation, you would have to know Assembler programming language, but the Dynamic Allocation feature has eliminated this requirement and has greatly simplified the process for allocating files. Your installation, however, must be a z/OS operating environment because Dynamic Allocation is designed to run above the 16-megabyte memory line.

While Dynamic Allocation can be used independently from IBM's APPC/MVS (Advanced Program-to-Program Communication for z/OS), using APPC/MVS in conjunction with Dynamic Allocation makes it possible for an application in use at one site to access an application at another site without having to physically change the JCL at the second site. For more information on APPC/MVS, see the *Advantage VISION:Results for z/OS Toolkit Reference Guide*.

Dynamic Allocation is implemented by the ALLOCATE command, using the following syntax:

```
ALLOCATE [FILE | FILES] [DYNAMICALLY] dataname
[STATUSFLAG dataname]
```

Figure 202 ALLOCATE Command Syntax

The ALLOCATE data name must be defined in the program syntax before the ALLOCATE command is executed. This data name contains the JCL statements that are used in dynamically allocating one or more files. The JCL must appear in 80-character card image format (therefore, the size of the ALLOCATE data name must be a multiple of 80) and must conform to JCL syntax rules. The syntax checker operates similarly to JES, including how it handles comments.

In addition, the optional keyword DYNAM can be added to the FILE statement:

```
FILE filename [DYNAM]
```

Figure 203 Keyword DYNAM in the FILE Statement

When the keyword DYNAM is coded on the FILE statement, VISION:Results will make certain that the program contains an explicit READ or WRITE for that file.

Rules for Dynamically Allocating Files

- The files must be explicitly read or written using the READ or WRITE commands. Dynamically allocated files cannot be handled by the automatic cycle and any attempt to do so may cause the program to abend.

- For a given dynamically allocated file, the ALLOCATE command must be executed before the first READ or WRITE is executed against that file.
- When the keyword DYNAM is coded on the FILE statement, VISION:Results will make certain that the program contains an explicit READ or WRITE for that file. A validation error occurs if no READ or WRITE is found.
- If used, the STATUSFLAG data name must be previously defined as a 1-byte character field. A Y, N, or F is placed in the STATUSFLAG data name field by the ALLOCATE command. Normally, all JCL syntax and Dynamic Allocation errors (message prefixes SVC99 and DYL1026E through 1074E) are written to SYSPRINT, but this can be overridden by placing an X in the STATUSFLAG data name field before executing the ALLOCATE command.

No return code is set by VISION:Results if a JCL syntax error or Dynamic Allocation failure is detected. To pass a condition code back to z/OS, query the STATUSFLAG after the ALLOCATE, move a value to DYLRRETURN, and issue a [STOP](#).

A status value of Y indicates that Dynamic Allocation was successfully completed. A value of N indicates that a JCL syntax error was detected and processing was stopped, and a value of F indicates that processing was completed but that z/OS reported a Dynamic Allocation failure.

When dynamically allocating VSAM KSDS RLS files, the keyword KEYLEN must be coded on the FILE statement. If the key position is not position 1 in the file, then the KEYLOC keyword must also be coded on the FILE statement.

- The JCL must be placed in columns 1 to 71 of the 80-character records; columns 72 to 80 will be ignored. VISION:Results determines the number of JCL lines (based on the size of the ALLOCATE data name) being allocated. The first blank record (columns 1 to 71) encountered terminates the JCL scanner, which ignores any JCL statements that follow.

JCL Considerations

Dynamic Allocation is only valid for JCL DD statements; however, comments are allowed. All JCL DD statements valid for JES are also honored by IBM's Dynamic Allocation routine (SVC 99) with the following exceptions:

- DSORG=VSAM is allowed (for SMS data sets only).
- Ddnames of JOBCAT, STEPCAT, JOBLIB, and STEPLIB are not allowed.
- Keyword parameters of AMP=, CHKPT=, DDNAME=, DLM=, and DSID= are not allowed.
- Ddnames that include stepnames are not allowed.
- Positional parameters of *, DATA, and DYNAM are not allowed.

- The following keyword and subparameter combinations are not allowed.

Keyword	Subparameter or Value
DCB	Reference to ddname of a previous step
	CYLOFL
	NTM
	RKP
DISP	PASS
DSN	Reference to ddname (*.ddname)
SPACE	ABSTR
UNIT	AFF
	SEP
VOLUME	RETAIN. If coded, it is not flagged as an error, but is ignored instead.
	REF=ddname
MSVGP	ddname specification. If coded, it is not flagged as an error, but is ignored instead.

Dynamic Allocation cannot allocate ddnames that are already allocated. All ddnames being used, such as SYSIN and SYSPRINT, are invalid for Dynamic Allocation.

Any JCL DD statements to be allocated must be valid statements according to JES syntax. To test the syntax of your JCL DD statements, run them in a batch job such as IEFBR14, as shown below:

```
// (user defined JOB card)
// STEP01 EXEC PGM=IEFBR14
// OUTFILE DD xxxxxxxx
.
.       These are the JCL statements
.       to be allocated.
.
```

Figure 204 Batch Job IEFBR14 Example

Executing the ALLOCATE command should produce the same results and error information as the test job shown in [Figure 204](#) (subject to the Dynamic Allocation exceptions noted previously).

If the ALLOCATE data name contains an initial blank record, a value of N is placed in the STATUSFLAG data name and a DYL-1027E message is generated.

If your VISION:Results program dynamically populates the ALLOCATE data name (see [Example 2—Allocate Facility and MOVE Command in OUTJCL Fields](#)), check that the first record of the data name is not blank before issuing the ALLOCATE command.

If you encounter any problems in getting the ALLOCATE command to execute properly, have a verified batch test run available before you contact Technical Support at <http://ca.com/support>, as well as a VISION:Results run with the STATUSFLAG not set to X.

You might also find it helpful, when new JCL is being allocated, to implement the STATUSFLAG field and check it through the VISION:Results program.

Procedural Command Examples

Example 1—Allocate Facility

This example contains only those VISION:Results statements required to illustrate the use of the ALLOCATE facility. It is not a complete VISION:Results program.

In Example 1, the JCL required for Dynamic Allocation is hard-coded, but the OUTJCL fields could have been populated by a MOVE command (or any other mechanism) as long as it was accomplished before the first explicit WRITE was done. This is illustrated in [Example 2—Allocate Facility and MOVE Command in OUTJCL Fields on page 321](#).

```
FILE OUTFILE OUTPUT FB 100 FROM OUTFILE DYNAM
OUTREC          100

FILE ININ DUMMY

WORKAREA
  JCLSTAT          1
  MSG1 41 VALUE 'SYNTAX ERROR IN DYNAMICALLY ALLOCATED'
           'JCL'
  MSG2 47 VALUE 'ALLOCATION FAILURE IN DYNAMICALLY'
           'ALLOCATED JCL'

WORKAREA
  OUTJCL          320      1
  FILLER          80      1
  VALUE '//OUTFILE DD DSN=DYL.BCM.OUTFILE,'
  FILLER          80
  VALUE '// DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,'
  FILLER          80
  VALUE '// SPACE=(TRK,(4,2),RLSE),'
  FILLER          80
  VALUE '// DCB=(RECFM=FB,BLKSIZE=2000,LRECL=100)'

ON ONE
  ALLOCATE FILE DYNAMICALLY OUTJCL STATUSFLAG JCLSTAT
  IF JCLSTAT EQ 'N'
    PRINT MSG1
    STOP
  ENDIF
  IF JCLSTAT EQ 'F'
```

Figure 205 Example 1. The Allocate Facility (incomplete program) (Page 1 of 2)


```

        PRINT EMSG2
        STOP
    ENDIF
ENDONE
.
.
; move data to OUTREC
.
.
WRITE OUTFILE

```

Figure 205 Example 1. The Allocate Facility (incomplete program) (Page 2 of 2)

Example 2—Allocate Facility and MOVE Command in OUTJCL Fields

This example contains only those VISION:Results statements required to illustrate the use of the ALLOCATE facility. It is not a complete VISION:Results program.

```

FILE OUTFILE OUTPUT FB 100 FROM OUTFILE DYNAM
   OUTREC          100

FILE ININ DUMMY

FILE JCLFILE FB 80 STATUS EOJCL
   JCLREC          80

;JCLFILE contains the JCL statements shown in the previous example

WORKAREA
   JCLSTAT          1
   MSG1 34 VALUE 'ERROR IN DYNAMICALLY ALLOCATED JCL'
   MSG2 38 VALUE 'NO JCL STATEMENTS IN ALLOCATE DATANAME'

WORKAREA
   OUTJCL 1600 VALUE NULL      ;initialize to blanks
   ALLOREC 80 1

ON ONE
RDJCL:
   READ JCLFILE
   IF EOJCL EQ 'E' GOTO ALCJCL   ENDIF
   MOVE JCLREC TO ALLOREC (INX) ;move 1 JCL record
   INX = INX + 80                ;set to move next JCL
                                   ;record
   GOTO RDJCL

ALCJCL:
   IF ALLOREC EQ BLANKS
       PRINT EMSG2
       STOP
   ENDIF
   ALLOCATE OUTJCL STATUSFLAG JCLSTAT
   IF JCLSTAT NE 'Y'
       PRINT MSG1
       STOP
   ENDIF
ENDONE
.
.
;move data to OUTREC
WRITE OUTFILE
.
.
.

```

Figure 206 Example 2. Allocate Facility and MOVE Command in OUTJCL Fields

Other Commands Affected by Dynamic Allocation

The following commands are also affected by Dynamic Allocation:

- [PICNSAVE Command](#)
- [REPORT Statement](#)
- [LTH Statement](#)
- [CALL Command](#)

PICNSAVE Command

A PICNSAVE file can be dynamically allocated:

```
FILE ININ DUMMY
WORKAREA
  FIELD1      50
  FIELD2     300
  FIELD3      30
WORKAREA
  OUTJCL      240    1
  FILLER      80    1
  VALUE ' //PICFILE DD DSN=DYL.BCM.PICFILE,'
  FILLER      80
  VALUE ' // DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,'
  FILLER      80
  VALUE ' // SPACE=(TRK,(8,4),RLSE)'
WORKAREA
  JCLSTAT      1
  ERRMSG 34 VALUE 'ERROR IN DYNAMICALLY ALLOCATED JCL'
ON ONE
  ALLOCATE OUTJCL STATUSFLAG JCLSTAT
  IF JCLSTAT NE 'Y'
    PRINT ERRMSG
    STOP
  ENDIF
ENDONE
PICNSAVE PICFILE USING FIELD1 FIELD2 FIELD3
.
```

Figure 207 PICNSAVE File Example

REPORT Statement

You can direct reports to a file other than SYS280R. These files are identified by the keyword SYS280Rx in the REPORT statement, as follows:

```
REPORT 160 WIDE 56 LONG SYS280Rx
```

You can apply Dynamic Allocation to the SYS280Rx file as long as the ALLOCATE is done before the first LIST command is executed for the report.

LTH Statement

You can direct letters to a file other than AUDPRINT. These files are identified by the LTH statement, as follows:

```
LTH 1 56 WIDE 50 LONG DYLTRxx BY  
10
```

You can apply Dynamic Allocation to the DYLTRxx file as long as the ALLOCATE is done before the first LETTER command is executed for the letter.

CALL Command

You can dynamically allocate files to be used by other programs (written in Assembler, FORTRAN, COBOL, and PL/1), which are accessed from VISION:Results using the CALL command.

Diagnostic Output and Error Messages

The ALLOCATE command performs validation during both the compile and execution phases of the VISION:Results program.

The compile phase validation error messages DYL-1075E through DYL-1081E relate to VISION:Results syntax errors. For more information, see *Advantage VISION:Results for z/OS Messages and Codes*.

The execution phase output and error messages relate to the syntax of the JCL statements being allocated and to the results of the Dynamic Allocation (SVC 99) from z/OS. The JCL syntax error messages are of the same nature as those found in JES. The Dynamic Allocation messages are the same messages found in the IBM manuals unless otherwise specified.

The execution phase generates the following:

- JCL images before any SVC 99 allocation
and
JCL images during SVC 99 allocation with the corresponding Dynamic Allocation message following.
- Syntax error messages in the DYL-1026E through DYL-1074E range (listed in the Error Messages manual). These errors can occur before any SVC 99 allocation or during the SVC 99 allocation process.
- Messages from z/OS Dynamic Allocation in the following form:

```
SVC99 xxxx-xxxx reason text. information text.
```

The xxxx-xxxx is the IBM-generated reason and information code respectively, followed by their corresponding IBM messages.

The execution phase information prints on SYSPRINT unless the ALLOCATE command specified a STATUSFLAG of X. The z/OS Dynamic Allocation failure messages always prints in the JES job log area, preceded by the statement: DYNAMIC ALLOCATION FAILURE! ERROR REASON CODE AND MESSAGE FOLLOW.

During the execution phase, two passes are made through the JCL. During the first pass, VISION:Results prints the JCL and checks the syntax for:

- Invalid JCL formats (for example, // missing in columns 1 and 2).
- Improper ddnames (for example, STEPNAME.DDNAME).
- Improper operation fields (for example, // DDNAME OUTPUT CLASS=A).

If a JCL syntax error occurs, syntax checking and printing of further JCL in the ALLOCATE data name is terminated.

During the second pass, VISION:Results prints the JCL again and checks the syntax for:

- Invalid JCL format (for example, a line ending with a comma followed by a new DD statement).
- Invalid keyword parameters (for example, DSNAMES=).
- Improper keyword parameters (for example, DDNAME=).
- Invalid parameter values (for example, DISP=(NOW,CATLG)).
- Improper parameter values (for example, VOL=REF=*.DDNAME).

If a JCL syntax error occurs, any further JCL processing terminates.

If a JCL syntax error occurs in either phase, the STATUSFLAG (if present) is set to N. If no STATUSFLAG is present, the program terminates with a U008 abend code.

If a DD statement is found to be valid, it is dynamically allocated (SVC 99). The resulting allocation message prints, as long as the STATUSFLAG has not been set to X. If an SVC 99 error is detected, the STATUSFLAG (if present) is set to F but JCL processing and allocation continue.

If no STATUSFLAG is present, JCL processing and allocation continue with possibly unpredictable results (for example, U300 abend) when the VISION:Results program attempts to access a file that may not have been properly allocated.

Chapter 15: Using the COPY Commands

VISION:Results program statements can reside in external data sets of various formats and, using the appropriate COPY command, be retrieved and copied into VISION:Results programs to reduce the time required to develop an application. Source can be stored on a partitioned data set (PDS), a source statement library (SSL), CONDOR, AllFusion® CA-Librarian®, AllFusion Endeavor, or AllFusion® CA-Panvalet® libraries or a DB2 table and be copied into VISION:Results programs by using the following COPY commands:

Command	Source Format	Operating System
COPY or COPYE	PDS	z/OS
COPY or COPYE	SSL	VSE
COPYC	CONDOR	VSE
COPYDB2	DB2	z/OS
COPYL	CA-Librarian	z/OS and VSE
COPYN	Endevor	z/OS
COPYP	CA-Panvalet	z/OS and VSE

Note: The COPY and COPYE commands are synonymous.

In this section, the generic term COPYx refers collectively to the COPY, COPYE, COPYC, COPYL, COPYN, and COPYP commands. COPYDB2 is described separately at the end of this section.

The COPY or COPYE macro facility allows you to keep frequently used applications in a library and yet enables you to dynamically change variables in the statements at execution time. This gives you the flexibility of designing your own specialized COPYx macro routines. A single COPYx statement can invoke an entire application from the appropriate library. With adequate foresight and analysis, you can use the COPYx facility to greatly reduce the amount of coding necessary in your program.

You have the option of using symbolic variables (definitions) or conditional generation statements in your stored copy code. The symbolic variables are identified by a # sign followed by a number or data name. Through the use of the COPYx command, the symbolic variables can be substituted for real values prior to inclusion into the program's input stream. In addition, COPYx supports COBOL data definitions as shown in [COPYX Command—Copies a Member or Book into VISION:Results on page 326](#).

If you want to include COBOL data definitions in your program without the use of COPYx, insert the command \$COBOL, followed by the data definitions, followed by the command \$ECOBOL (in column 8 or later). For more information about the COBOL support facility, see [Using DB2 Tables on page 345](#).

If you have a DB2 subsystem installed, you can copy the names of a DB2 table into VISION:Results to be used as field definitions. For more information, see [Using COBOL Record Descriptions on page 340](#).

Formats

COPYX Command—Copies a Member or Book into VISION:Results

Use the COPYx command to copy a z/OS member or a VSE book into the VISION:Results program.

This command has the following format:

```
COPYx  membername [! [{#A=} ] [{dataname}]!...] [COBOL] [REPLACING string1 BY string2]]
      [ [ [{#n=} ] [ {literal } ] ]
      [ {NULL } ] ]
```

COPYx

Identifies which COPYx command will be used. This command may be COPY, COPYC, COPYE, COPYL, COPYN, or COPYP.

member

For z/OS, specifies the name of the member of the PDS, CA-Librarian, CA-Panvalet member, or Endeavor element. For VSE, specifies the member type and book name to be copied into the VISION:Results program. If VSE, the member parameter immediately following the COPYx command is the member type under which the statements (member or book) were cataloged and book name. The book name of the statements to be inserted follows the member type and period if the default member type is not used: membertype.bookname. VISION:Results default for the member type is D.

COBOL

Indicates that the retrieved item is a COBOL record or working storage description. The member name (z/OS) or book name (VSE) is retrieved from the appropriate library, listed, and included as part of your program. Any field in your COBOL description can then be referenced elsewhere in your VISION:Results program. A default column heading equal to the data name itself is established for each field. For more information about this command format, see [Using COBOL Record Descriptions on page 340](#).

Symbolic Variables

Up to 36 symbolic variables (#0 to #9, #A to #Z) can be specified following the member name, optionally followed by a substitution value. These are referred to as overrides. Continuation lines to define additional symbolic variables are permissible. There is no limit on the number of continuation lines. The maximum value allowed for a symbolic variable is 30 continuous characters.

#Y and #Z have automatic default values and, if used, must only be specified as overrides in the COPYx statement itself. #Y default is O for z/OS; D for VSE. #Z is a counter containing the number of COPYx statements invoked (first COPY is 01).

The order in which symbolic variables are defined on the COPYx or \$DEFAULT statements is not relevant. #2 can precede #1, #W can precede #B, etc.

Example of Permissible Symbolic Values	Explanation of Value
#1=DATANAME	Value is as specified.
#1=DATA_NAME	Value is as specified.
#1=','	Comma delimited by single quotation marks.
#2='b'	Blank delimited by single quotation marks.
#5="X'FF'"	Single quotation marks delimited by double quotation marks. Value extracted is X'FF.'
#A='b b 5'	Blanks and 5 delimited by single quotation marks.
#4="C'O'D'E"	Single quotation marks delimited by double quotation marks.
#3='V5b b b'	Blanks delimited by single quotation marks.
#C='"'	Double quotation marks delimited by single quotation marks.
#Y='VALUEb , ISb b b , b b'	Blanks and commas delimited by single quotation marks.
#Z=ABCD	Value is as specified.

Example of Permissible Symbolic Values	Explanation of Value
#0=REJECT	Value is as specified.
#3=ACCEPT	Value is as specified.
#Q=NULL	NULL conditionally includes or excludes code.
	No value.

z/OS COPYx Syntax Examples

```

COPY ARFILE
COPY PAYROLL COBOL
COPY ARFILE
COPY ARFILE #1=REJECT #2=ACCEPT
COPY PAYROLL #B=ACCEPT #A=NULL
COPY APMSTR #2='REPORT-3A' #1='12/03/89'
          #0= #4=DATE #5=AMOUNT

```

Note: COPY can also be COPYE, COPYL, COPYN, or COPYP.

VSE COPY Syntax Examples

```

COPY D.ARSORT #1=REJECT #2=ABC
COPY ARSORT #1=REJECT #2=ABC      (member type default is taken)
COPY D.ARFILE #1= #2= #3=NULL
COPY Q.PAYROLL #A='100,000.00' #B='REMARKS' #6=NEXT
COPY APMSTR #0=BALANCE #1=REJECT #2=ACCEPT (member type default is taken)

```

Note: COPY can also be COPYC, COPYE, COPYL, or COPYP.

The member type default in VISION:Results is D.

\$DEFAULT and \$IF

You can use the \$DEFAULT and the \$IF commands when symbolic variable values are used. These commands are coded and stored in the item that is retrieved. Unlike the other retrieved commands that display in your program, \$DEFAULT and \$IF reflect the commands after the parameter substitution has been made or condition has been executed.

\$DEFAULT gives default values to symbolic variables that are not found in the COPYx statement as overrides. The \$DEFAULT statement must be coded as the first statement in any block of code that is to become a member or book that will be retrieved. Following the keyword \$DEFAULT, up to 36 symbolic variables (#0 - #9, #A - #Z) can be specified. The \$DEFAULT block is delimited by the \$DEND command which occurs on a line by itself. It cannot contain any other commands, keywords, or extraneous data on the same input line.

Overriding \$DEFAULT values is allowed, but they must be specified on the COPYx statement following the member/book name or in a continuation line.


```

$DEFAULT #1=REJECT #2=ACCEPT
#3=STOP #4=NEXT #5='2,000.00'
$DEND
IF ACCOUNT EQ 'WO' #1  ENDIF

```

\$IF is the conditional statement facility of the COPYx command (Format 2). The inclusion or exclusion of statements is completely dependent upon the symbolic variable, immediately following the **\$IF**, having any value or being NULL.

\$IF must be followed by at least one space and a symbolic variable (**#?**), where ? is 0 through 9 or A through Z. No other data can be specified on this input line.

\$IF must have a corresponding **\$IFE** as its last record to terminate the inclusion or exclusion of source code. No other data can be specified on this input line.

There is no limit on the number of **\$IF** and **\$IFE** pairs; however, they cannot be embedded within one another. One set must be completed before a second set can be invoked.

\$IF and **\$IFE** do not print as copy code.

If the **\$IF** symbolic variable (**#?**) has an override or default value, all source immediately following it up to **\$IFE** is included. If the **\$IF** symbolic variable (**#?**) value is null, all of the source immediately following it up to **\$IFE** is not included.

Example 1

```

$DEFAULT #1=NULL
$DEND
$IF #1
    SORT ARFILE USING #2 #3
$IFE

```

If **#1** is given any value on the COPYx statement, the SORT statement is included in the input stream. If the default is taken (no override in the COPYx statement), the statement is not included.

Example 2

```

$DEFAULT #1=YES
$DEND
$IF #1
    SORT ARFILE USING #2 #3
$IFE

```

In Example 2, the SORT is included whether **#1** is given a value or not. However, if the value given is specified as NULL (**#1=NULL**), the SORT statement is not included.

The **\$ELSE** clause allows you to include alternate logic in a **\$IF** statement. The **\$ELSE** clause must be on a line by itself and must be preceded by a valid **\$IF** statement.

```

$IF #A
    MOVE TEMPSTAT TO OUTFILEB.STAT
    WRITE OUTFILEB

```

```
$ELSE
  MOVE REALSTAT TO OUTFILEA.STAT
  WRITE OUTFILEA
$IFE
```

In the above example, if the #A symbolic variable is defined and not null, the first MOVE and WRITE statements are executed. If #A is undefined or null, the \$ELSE logic is used.

Examples of Source Code

The following examples show:

- The source as it might appear as a member of a PDS or as a VSE source library.
- How to start the z/OS command. VSE is identical with the exception of the member type and period.
- The resulting statement stream after the COPYx command has been encountered.

Example 1 COPY with No Symbolics

DYAUDREC Contents

ACCTNO	7	4					(ACCOUNT'NUMBER)
TRANDATE	6	38					(TRANSACTION'DATE)
BILLNGDATE	6	44	NU	D			(BILLING'DATE)
NAMADR	75	85					(NAME AND ADDRESS)
NAME	25	85					
ADD1	25	110					(STREET)
ADD2	25	135					(STATE'ZIP)
BALANCE	5	170	PD	2	A		
ACCOUNT	2	182					(ACCOUNT'CODE)
INSTLBAL	6	191	PD	2	A		(INSTALL'BALANCE)
INSTPAY	5	197	PD	2	A		(INSTALL'PAY)
BALPART	4	202	PD	2	A		(BALANCE PART'PAY)
INTPART	3	206	PD	2	A		(INTEREST PART'PAY)
NUMPY	2	209	PD	A			(NUMBER'PAY)

Figure 208 [Example 1](#) COPY with No Symbolics, DYAUDREC Contents

z/OS Input Stream

```
FILE ARFILE FB 352 5280
COPY DYAUDREC
SORT ARFILE USING ACCOUNT
LIST ACCOUNT TRANS BALANCE
T1 'REPORT 1'
```

Figure 209 [Example 1](#) COPY with No Symbolics, z/OS Input Stream

Final VISION:Results Program

```

FILE ARFILE FB 352 5280
COPY DYAUDREC
COPIED ACCTNO      7      4      (ACCOUNT'NUMBER)
COPIED TRANDATE    6      38     (TRANSACTION'DATE)
COPIED BILLNGDATE  6      44     (BILLING'DATE)
COPIED NAMADR      75     85     (NAME AND ADDRESS)
COPIED NAME        25     85
COPIED ADD1        25     110    (STREET)
COPIED ADD2        25     135    (STATE'ZIP)
COPIED BALANCE     5     170    PD  2  A
COPIED ACCOUNT     2     182    (ACCOUNT'CODE)
COPIED INSTLBAL    6     191    PD  2  A (INSTALL'BALANCE)
COPIED INSTPAY     5     197    PD  2  A (INSTALL'PAY)
COPIED BALPART     4     202    PD  2  A (BALANCE PART'PAY)
COPIED INTPART     3     206    PD  2  A (INTEREST PART'PAY)
COPIED NUPY        2     209    PD  2  (NUMBER'PAY)
SORT ARFILE USING ACCOUNT
LIST ACCOUNT TRANS BALANCE
T1 'REPORT 1'

```

Figure 210 [Example 1](#) COPY with No Symbolics, Final VISION:Results Program

Example 2 COPY with Symbolics, No Defaults**SORTDEF Contents**

```

SORT #1 USING #2

```

Figure 211 [Example 2](#) COPY with Symbolics, No Defaults, SORTDEF Contents

z/OS Input Stream

```

FILE ARFILE FB 352 5280
ACCOUNT 2 182
TRANS 7 4
BALANCE 5 170 PD 2 A
COPY SORTDEF #1=ARFILE #2=ACCOUNT
LIST ACCOUNT TRANS BALANCE
T1 'REPORT 2'

```

Figure 212 [Example 2](#) COPY with Symbolics, No Defaults, z/OS Input Stream

Final VISION:Results Program

```

FILE ARFILE
ACCOUNT 2 182
TRANS 7 4
BALANCE 5 170 PD 2 A
COPY SORTDEF #1=ARFILE #2=ACCOUNT
COPIED SORT ARFILE USING ACCOUNT
LIST ACCOUNT TRANS BALANCE
T1 'REPORT 2'

```

Figure 213 [Example 2](#) COPY with Symbolics, No Defaults, Final VISION:Results Program

Example 3 COPY with Symbolics and Defaults

SORTDEF Contents

```
$DEFAULT #1=ARFILE #2=ACCOUNT
          #3=TRANS
$DEND
  SORT #1 USING #2 #3
```

Figure 214 [Example 3](#) COPY with Symbolics and Defaults, SORTDEF Contents

z/OS Input Stream

```
FILE ARFILE FB 352 5280
  ACCOUNT 2 182
  TRANS 7 4
  BALANCE 5 170 PD 2 A
COPY SORTDEF
LIST ACCOUNT TRANS BALANCE
T1 'REPORT 3'
```

Figure 215 [Example 3](#) COPY with Symbolics and Defaults, z/OS Input Stream

Final VISION:Results Program

```
FILE ARFILE FB 352 5280
  ACCOUNT 2 182
  TRANS 7 4
  BALANCE 5 170 PD 2 A
COPY SORTDEF
COPIED $DEFAULT #1=ARFILE #2=ACCOUNT
COPIED          #3=TRANS
COPIED $DEND
COPIED SORT ARFILE USING ACCOUNT TRANS
COPIED LIST ACCOUNT TRANS BALANCE
COPIED T1 'REPORT 3'
```

Figure 216 [Example 3](#) COPY with Symbolics and Defaults, Final VISION:Results Program

Example 4 COPY with Symbolics, Defaults, and Partial Default Override

SORTDEF Contents

```
$DEFAULT #1=ARFILE #2=ACCOUNT
          #3=TRANS
$DEND
  SORT #1 USING #2 #3
```

Figure 217 [Example 4](#) COPY with Symbolics, Defaults, and Partial Default Override, SORTDEF Contents

z/OS Input Stream

```

FILE ARFILE FB 352 5280
  ACCOUNT 2 182
  TRANS 7 4
  BALANCE 5 170 PD 2 A
COPY SORTDEF #3=NAME
LIST ACCOUNT TRANS BALANCE
T1 'REPORT 4'

```

Figure 218 [Example 4](#) COPY with Symbolics, Defaults, and Partial Default Override, z/OS Input Stream

Final VISION:Results Program

```

FILE ARFILE FB 352 5280
  ACCOUNT 2 182
  TRANS 7 4
  BALANCE 5 170 PD 2 A
COPY SORTDEF #3=NAME
COPIED $DEFAULT #1=ARFILE #2=ACCOUNT
COPIED #3=TRANS
COPIED $DEND
COPIED SORT ARFILE USING ACCOUNT NAME
LIST ACCOUNT TRANS BALANCE
T1 'REPORT 4'

```

Figure 219 [Example 4](#) COPY with Symbolics, Defaults, and Partial Default Override, Final VISION:Results Program

Example 5 COPY with Symbolics, Defaults, and Use of Two COPY Statements

LISTAR Contents

```

$DEFAULT #1=ACCOUNT
          #2=TRANS
          #3=NAME
          #4=BALANCE
          #5=BILLNGDATE
          #6='SUM NUMPY'
$DEND
CONTROL #1
LIST #1 #2 #3 #4 #5
ON CHANGE #1
  LIST SUM #4 #6 WITH 2 BEFORE

```

Figure 220 [Example 5](#) COPY with Symbolics, Defaults, and Use of Two COPY Statements, LISTAR Contents

z/OS Input Stream

```

FILE ARFILE FB 352 5280
COPY DYAUDREC
SORT ARFILE USING ACCOUNT
COPY LISTAR #5=INTPART
#4=INSTLBAL #6=TALLY
T1 'REPORT 5' WITH 2 AFTER

```

Figure 221 [Example 5](#) COPY with Symbolics, Default, and Use of Two COPY Statements, z/OS Input Stream

Final VISION:Results Program

```
FILE ARFILE FB 352 5280
COPY DYAUDREC
COPIED ACCTNO      7      4      (ACCOUNT'NUMBER)
COPIED TRANDATE    6      38     (TRANSACTION'DATE)
COPIED BILLNGDATE  6      44     (BILLING'DATE)
COPIED NAMADR      75     85     (NAME AND ADDRESS)
COPIED NAME        25     85
COPIED ADD1        25     110    (STREET)
COPIED ADD2        25     135    (STATE'ZIP)
COPIED BALANCE     5     170 PD  2  A
COPIED ACCOUNT     2     182    (ACCOUNT'CODE)
COPIED INSTLBAL    6     191 PD  2  A (INSTALL'BALANCE)
COPIED INSTPAY     5     197 PD  2  A (INSTALL'PAY)
COPIED BALPART     4     202 PD  2  A (BALANCE PART'PAY)
COPIED INTPART     3     206 PD  2  A (INTEREST PART'PAY)
COPIED NUMPY       2     209 PD  A   (NUMBER'PAY)
SORT ARFILE USING ACCOUNT
COPY LISTAR #5=INTPART
#4=INSTLBAL #6=TALLY
COPIED $DEFAULT    #1=ACCOUNT
COPIED              #2=TRANS
COPIED              #3=NAME
COPIED              #4=BALANCE
COPIED              #5=BILLNGDATE
COPIED              #6='SUM NUMPY'
COPIED $DEND
COPIED CONTROL ACCOUNT
COPIED LIST ACCOUNT TRANS NAME INSTLBAL INTPART
COPIED ON CHANGE ACCOUNT
COPIED LIST SUM INSTLBAL TALLY WITH 2 BEFORE
COPIED T1 'REPORT 5' WITH 2 AFTER
```

Figure 222 Example 5 COPY with Symbolics, Defaults, and Use of Two COPY Statements, Final VISION:Results Program

Example 6 COPY Conditional Statement**ARSORT Contents**

```
$DEFAULT #1=ARFILE #2=ACCOUNT
#3=TRANS #4=YES #5=NULL
#6=NULL
$DEND
$IF #4
  SORT #1 USING #2 #3
$IFE
$IF #5
  SORT #1 USING #2
$IFE
$IF #6
  IF TRANS EQ '3330000' THRU '6890000' REJECT ENDIF
  SORT #1 USING #3
$IFE
```

Figure 223 [Example 6](#) COPY Conditional Statement, ARSORT Contents

z/OS Input Stream

```

FILE ARFILE FB 352 5280
  ACCOUNT 2 182
  TRANS 7 4
  BALANCE 5 170 PD 2 A
COPY ARSORT #4=NULL #5=YES
LIST ACCOUNT TRANS BALANCE
T1 'REPORT 6'

```

Figure 224 [Example 6](#) COPY Conditional Statement, z/OS Input Stream

Final VISION:Results Program

```

FILE ARFILE FB 352 5280
  ACCOUNT 2 182
  TRANS 7 4
  BALANCE 5 170 PD 2 A
COPY ARSORT #4=NULL #5=YES
COPIED $DEFAULT #1=ARFILE #2=ACCOUNT
COPIED #3=TRANS #4=YES #5=NULL
COPIED #6=NULL
COPIED $DEND
COPIED SORT ARFILE USING ACCOUNT
LIST ACCOUNT TRANS BALANCE
T1 'REPORT 6'

```

Figure 225 [Example 6](#) COPY Conditional Statement, Final VISION:Results Program

Example 7 COPY Conditional Statement

ARSORT Contents

```

$DEFAULT #1=ARFILE #2=ACCOUNT
#3=TRANS #4=YES #5=NULL
#6=NULL
$DEND
$IF #4
  SORT #1 USING #2 #3
$IFE
$IF #5
  SORT #1 USING #2
$IFE
$IF #6
  IF TRANS EQ '3330000' THRU '6890000' REJECT ENDIF
  SORT #1 USING #3
$IFE

```

Figure 226 [Example 7](#) COPY Conditional Statement, ARSORT Contents

z/OS Input Stream

```

FILE ARFILE FB 352 5280
  ACCOUNT 2 182
  TRANS 7 4
  BALANCE 5 170 PD 2 A
COPY ARSORT #4=NULL #6=YES
LIST TRANS ACCOUNT BALANCE
T1 'REPORT 7'

```

Figure 227 [Example 7](#) COPY Conditional Statement, z/OS Input Stream

Final VISION:Results Program

```
FILE ARFILE FB 352 5280
      ACCOUNT 2 182
      TRANS   7   4
      BALANCE 5 170 PD 2 A
      COPY ARSORT #4=NULL #6=YES
COPIED $DEFAULT #1=ARFILE #2=ACCOUNT
COPIED      #3=TRANS #4=YES #5=NULL
COPIED      #6=NULL
COPIED $DEND
COPIED IF TRANS EQ '3330000' THRU '0890000' REJECT ENDIF
COPIED SORT ARFILE USING TRANS
      LIST TRANS ACCOUNT BALANCE
      T1 'REPORT 7'
```

Figure 228 [Example 7](#) COPY Conditional Statement, Final VISION:Results Program

General Considerations

The following considerations apply to the COPYx commands:

- There is no limit on the number of COPYx commands used.
- COPYx commands cannot be embedded within one another. (A COPYx cannot invoke another COPYx.)
- Spaces and commas are delimiters. To include special characters such as spaces, commas, or quotation marks in a literal string, they must be enclosed in quotation marks. Single quotation marks must be enclosed within double quotation marks ("). Double quotation marks must be enclosed within single quotation marks (').
- Each value, unless enclosed within single or double quotation marks, must be a continuous string.
- Values cannot continue onto a second statement. The value cannot begin and end on two separate lines.
- If the keyword CARD or CARDS is used in a FILE statement to indicate instream data, the FIN statement that precedes the data cannot be input to VISION:Results using the COPYx command.
- Nulls can be specified in the \$DEFAULT, \$IF, and COPYx commands. Nulls can be specified in two ways, #1= or #1=NULL. However, if the symbolic variable is to be defined as a null and it is the last symbolic variable specified on a COPYx, it must be specified with the keyword NULL (#1=NULL).

```
COPY ARSORT #1=REJECT #2=REJECT #3=NULL
COPY APSELECT #1= #2= #3= #4=NULL
COPY APSELECT #1=NULL #2=NULL #3=NULL #4=NULL
```


Source Format Considerations

This section provides guidelines for coding COPYx commands for z/OS and VSE.

COPY or COPYE Considerations—z/OS

An additional JCL SYSCOPY DD statement is required to specify the name of the partitioned data set (PDS) containing the VISION:Results statement members to be copied into the VISION:Results program input stream, as shown below:

```
//SYSCOPY DD DSN=pdsname,DISP=SHR
```

For information about creating a member of a PDS, see the IBM utility documentation.

If your system has CA-Panvalet or CA-Librarian installed as a SUBSYSTEM to z/OS, you may retrieve items of those systems as members of a PDS using COPY or COPYE if the following DD statement is used for SYSCOPY:

```
//SYSCOPY DD DSN=libname,DISP=SHR,SUBSYS=name
```

where libname is the name of your CA-Panvalet or CA-Librarian source library, and name is the SUBSYSTEM name established when your copy facility was installed.

Note: Depending upon their release, some APARs may be required by your CA-Panvalet or CA-Librarian system.

In these cases the CA-Librarian or CA-Panvalet system loadlib must be included in the JOBLIB or STEPLIB concatenation if they are not present in the linklist.

COPY or COPYE Considerations—VSE

Additional VSE JCL may be required if any of the statements (BOOKS) to be copied reside on a private source statement library (SSL). An example of the JCL required for the COPY or COPYE VISION:Results program statements contained on a private SSL is shown below:

```
// DLBL DYLLIB,'VISION:Results.Private.SSL'  
// EXTENT, VOL001 FOR VSE  
// LIBDEF SOURCE,SEARCH=DYLLIB.SSL
```

Note: For information about cataloging books to a source statement library, see the description of the LIBR program in the appropriate IBM publication.

The private source statement library, if assigned, is searched first for the specified member type book. If the book is not found, the system source statement library is then searched.

COPYC Considerations—VSE

COPYC retrieves members from your CONDOR library. The maximum member name length supported is eight characters. VISION:Results assumes a file name of GACLIB for the CONDOR library. It also assumes that the CONDOR library is on the same device type as SYSRES. Add a DLBL and EXTENT statement for the CONDOR library to the standard VISION:Results JCL before running a program that contains COPYC.

```
// DLBL GACLIB, '          '  
// EXTENT
```

COPYL Considerations

COPYL retrieves members from your CA-Librarian file. The maximum member name length supported is eight characters.

CA-Librarian z/OS JCL

A MASTER DD statement describing the CA-Librarian file is required.

```
//MASTER DD DSN=Your.LIBRARIAN.source.Library,DISP=SHR
```

If CA-Librarian was not statically linked with VISION:Results when VISION:Results was installed, then the CA-Librarian system loadlib must be included in the JOBLIB or STEPLIB concatenation. The dynamic link is supported only for CA-Librarian r3.0 and higher.

To retrieve the CA-Librarian item as a member of a PDS, see the [COPY or COPYE Considerations—z/OS on page 337](#) above.

CA-Librarian VSE JCL

DLBL and EXTENT statements are required to describe the CA-Librarian file. The file name is MASTER.

```
// DLBL MASTER, '          '  
// EXTENT
```

If CA-Librarian was not statically linked with VISION:Results when VISION:Results was installed, then the CA-Librarian system loadlib must be included in the JOBLIB or STEPLIB concatenation. The dynamic link is supported only for CA-Librarian r3.0 and higher.

COPYN Considerations

COPYN retrieves elements from your Endeavor managed library. COPYN is supported for Endeavor r7.1 and higher. The maximum element name length supported is 10 characters.

Endevor requires that users executing a non-authorized program, such as VISION:Results, must have the requisite security access to the Endevor data sets for the function requested. The caller's security profile must have sufficient eTrust™ CA-Top Secret® Security, eTrust™ CA-ACF2® Security, or RACF access to the Endevor control files (catalog, master, package, base, delta), because the Endevor alternate ID facility is not available when running in an unauthorized mode.

JCL modifications are required for your VISION:Results application with Endevor.

Add the Endevor CONLIB and AUTHLIB libraries to the STEPLIB or JOBLIB concatenation:

```
//STEPLIB DD DISP=SHR,DSN=your.endevor.AUTHLIB
//          DD DISP=SHR,DSN=your.endevor.CONLIB
```

Two additional DD statements are used: one is for a workfile to contain the retrieved element before it is inserted into the VISION:Results program, and the other is a message file to record messages generated by Endevor during the retrieval process. If coded by the user, the message file DDname must be either D\$NDVMSG or APIMSGS, and the workfile DDname must be D\$NDVCPY such as:

```
//D$NDVMSG DD SYSOUT=*
//D$NDVCPY DD UNIT=SYSDA,DISP=(,DELETE),DSN=&&WRKN,
//          SPACE=(TRK,(5,1)),RECFM=VB,LRECL=2048
```

If these DDnames are not included in the JCL, the system dynamically allocates a D\$NDVMSG file as SYSOUT=* and a D\$NDVCPY file with attributes RECFM=VB,LRECL=2048.

COPY Considerations

COPY retrieves members from your CA-Panvalet library. COPY support allows multiple AllFusion CA-Panvalet libraries to be searched when including source statements during compile. The maximum member name length supported is 10 characters.

CA-Panvalet z/OS JCL

A DD statement describing each CA-Panvalet source library is required. The ddname search sequence is PANDD1, then PANDD11 through PANDD19. PANDD11 through PANDD19 are optional.

```
//PANDD1 DD DSN=Your.PANVALET.source.Library,DISP=SHR
//PANDD11 DD DSN=Your.next.PANVALET.source.library,DISP=SHR
```

If CA-Panvalet was not statically linked with VISION:Results when VISION:Results was installed, then the CA-Panvalet system loadlib must be included in the JOBLIB or STEPLIB concatenation.

To retrieve the CA-Panvalet item as a member of a PDS, see the [COPY or COPYE Considerations—z/OS on page 337](#).

CA-Panvalet VSE JCL

DLBL and EXTENT statements are required to describe each CA-Panvalet library. The source file name is PANDD1.

```
// DLBL PANDD1, '          '  
// EXTENT
```

Using COBOL Record Descriptions

This section provides guidelines for using COBOL record descriptions.

Instream COBOL Facility

To include COBOL data definition statements directly (without the use of COPYx), insert the \$COBOL command ahead of the descriptions. Following the last COBOL statement, insert the command \$ECOBOL. The \$ECOBOL command must be coded in column 8 or later.

If you have a COBOL record or working storage description stored in a library, you can copy it directly into your VISION:Results program using the COPYx commands.

The stored item is retrieved from the appropriate library, listed, and included as part of your program. Any field in your COBOL description can then be referenced elsewhere in your VISION:Results program. A default column heading equal to the data name itself is established for each field.

Replacing Character Strings

You can replace one or more character strings by another character string. The syntax is as follows:

```
COPYx member COBOL REPLACING operand 1 BY operand 2
```

Figure 229 Replacing Character Strings Syntax

Each operand can be an alphanumeric literal or COBOL pseudo-text. An alphanumeric literal can be up to 30 contiguous alphanumeric characters. Blanks and special characters are not permitted unless the entire operand is enclosed in quotation marks. Pseudo-text must be enclosed in double equal signs. The pseudo-text string can be up to 30 characters long, not including the delimiting equal signs. If blanks or special characters are included in the pseudo text, the entire string, including the double equal sign delimiters, must be enclosed in quotation marks.

You can replace up to 30 characters of a data name. Substitution is made for the first occurrence of the string encountered on each line. You are responsible for assuring that the replaced text does not extend past the end of the line.

Rules for Using COBOL Record Descriptions

The following rules govern these facilities:

- The copied member or book must contain only valid COBOL record descriptions or working storage descriptions. An asterisk in column 7 (comment statement) is also permitted.
- All hyphens (-) in data names are changed internally to underscores (_). All single character data names have a trailing underscore added (for example, A_). Although your listing of the copied code does not show the underscores, you must use the underscore rather than the hyphen when referencing the data name.
- Numeric, packed, and binary fields cannot have more than 9 decimal places, nor more than 16 digits including decimal places. Decimal scaling positions (P) in the PICTURE clause are dropped.
- The VALUE clause is not allowed in an input-file description.
- Alignment:
 - Every 01 level item subordinate to FILE points to the first byte of the record area (implied redefinition).
 - 01 level and COMP-2 items are aligned on the next doubleword boundary.
 - COMP-1 items are aligned on the next fullword boundary.
 - Synchronized (SYNC) items are aligned on the next halfword or fullword boundary as required.
- Report items, floating-point items, and 8-byte binary items are changed to character fields having an equivalent length.
- Level 66 and level 88 items are dropped.
- The following clauses are ignored:
 - ASCENDING [KEY IS]
 - BLANK [WHEN] ZERO
 - COPY
 - DEPENDING [ON]
 - DESCENDING [KEY IS]
 - INDEX (USAGE IS)
 - INDEXED [BY]
 - JUST(IFIED) [RIGHT]
 - SIGN
 - TO (variable # of occurrences)
- Unless another edit code is specified in the LIST statement, all non-character fields are assigned an edit code of A. When the field is listed, it has high order zero suppression with commas and decimal insertion. This default edit code can be changed using the COBEDIT parameter of the DYLINSTL macro. For details, see the *Advantage VISION:Results for z/OS Installation Guide*.

- The COBOL 'SIGN IS LEADING SEPARATE' or 'SIGN IS TRAILING SEPARATE' clause on data definitions is supported when the DYLINSTL macro parameter CBXSIGN=Y or the OPTION COBXSIGN statement is used. The data element being processed will be defined as a character field with the length increased by 1 to include the sign position.
- COBOL VALUE clauses containing embedded double quotes for character fields will be stored with a single quote if DYLINSTL parameter COBAPOS is set to Y or OPTION parameter COBAPOS is specified.
- You can edit COBOL data fields by specifying the edit code immediately after the data name in your LIST statement. For example:

```
LIST A_ PO_SPL_QTY PO_SPL_DTE D PO_RCVR_DATA
```

The date field (PO_SPL_DTE) is printed with slashes (for example, 01/15/01).

- All COBOL record descriptions must have been previously compiled with the COBOL compiler to make sure they are valid. Invalid COBOL record descriptions that are processed by VISION:Results can cause unpredictable results.

VISION:Results also supports COBOL data names beginning with a numeric character. These are treated as valid data names and can be used by prefixing an N_ to the data name. For example:

```
05 120DATE PIC 9(6).          COBOL definition
MOVE N_120DATE TO OUTDATE     VISION:Results usage
```

COBOL Examples

This section contains COBOL examples.

COPY Member COBOL

```
FILE PURCHMST FB 245 2450
COPY POMSTR COBOL
LIST A_ PO_SPL_QTY PO_SPL_DTE PO_RCVR_DATA
```

Figure 230 COBOL Example, COPY Member COBOL

Program Listing

```
FILE PURCHMST FB 245 2450
COPY POMSTR COBOL
COPIED      01 PO-MSTR-LAYOUT.
COPIED      02 PO-REC-BODY      PIC X(171).
COPIED      02 A                PIC X(45).
COPIED      02 TRAILERS.
COPIED      03 PO-DELIVERIES.
COPIED      05 PO-SPL-QTY      PIC S9(5)    COMP-3.
COPIED      05 PO-SPL-DTE      PIC 9(07)    COMP-3.
COPIED      03 PO-RECEIVERS.
COPIED      05 PO-RCVR-DATA    PIC X(22).

LIST A_ PO_SPL_QTY PO_SPL_DTE PO_RCVR_DATA
```

Figure 231 COBOL Example, Program Listing 1

```

FILE PURCHMST FB 245 2450
COPY POMSTR COBOL
COPIED      01  PO-MSTR-LAYOUT.
COPIED      02  PO-REC-BODY      PIC X(171) .
COPIED      02  A                PIC X(45) .
COPIED      02  TRAILERS.
COPIED      03  PO-DELIVERIES.
COPIED      05  PO-SPL-QTY      PIC S9(5)    PACKED-DECIMAL.
COPIED      05  PO-SPL-DTE      PIC 9(07)    PACKED-DECIMAL.
COPIED      03  PO-RECEIVERS.
COPIED      05  PO-RCVR-DATA    PIC X(22) .

LIST A_     PO_SPL_QTY    PO_SPL_DTE    PO_RCVR_DATA

```

Figure 232 COBOL Example, Program Listing 2

The COBOL data description is printed in the VISION:Results program listing exactly as it was received. That is, data names are printed with hyphens and as one-character data names, just as they were originally coded. However, when these fields are referenced in a VISION:Results statement (for example, the LIST statement above), the data names must have an underscore (_) instead of the hyphen and a one-character data name must have an underscore appended (A_).

Instream COBOL

```

FILE PURCHMST FB 245 2450
$COBOL
  01  PO-MSTR-LAYOUT.
  02  PO-REC-BODY      PIC X(171) .
  02  A                PIC X(45) .
  02  TRAILERS.
  03  PO-DELIVERIES.
  05  PO-SPL-QTY      PIC S9(5)    COMP-3.
  05  PO-SPL-DTE      PIC 9(07)    COMP-3.
  03  PO-RECEIVERS.
  05  PO-RCVR-DATA    PIC X(22) .
$ECOBOL

LIST A_     PO_SPL_QTY    PO_SPL_DTE    PO_RCVR_DATA

```

Figure 233 COBOL Example, Instream COBOL 1

COBOL Replacing Literal

VISION:Results Statement

```
COPY SLSOR101 COBOL REPLACING XYZ BY 'C101-'
```

COBOL Copybook Member

```

01  XYZSLSOR101.
05  XYZSLSOR101-KEY.
    10  XYZ0-LOCN-ADMIN      PIC X(05) .
    10  XYZ0-WORK-ORD-NUM    PIC 9(10) .
05  XYZ0-CUST-NUM            PIC 9(07) .
05  XYZ0-CUST-BILL-TO        PIC X(07) .

```

Figure 234 COBOL Replacing Literal, COBOL Copybook Member

Copied Member in VISION:Results

```

01 C101-SLS04101-KEY.
05 C101-0-SLS0R101-KEY.
   10 C101-LOCN-ADMIN          PIC X(05).
   10 C101-WORK-ORD-NUM        PIC 9(10).
05 C101-CUST-NUM               PIC 9(07).
05 C101-CUST-BILL-TO           PIC X(07).

```

Figure 235 COBOL Replacing Literal, Copied Member in VISION:Results

COBOL Replacing Pseudo-Text

VISION:Results Statement

```
COPY SLSOR101 COBOL REPLACING '==:XYZ:==' BY '==C-01-=='
```

COBOL Copybook Member

```

01 :XYZ:SLSOR101.
05 :XYZ:SLSOR101-KEY.
   10 :XYZ:0-LOCN-ADMIN        PIC X(05).
   10 :XYZ:0-WORK-ORD-NUM      PIC 9(10).
05 :XYZ:0-CUST-NUM             PIC 9(07).
05 :XYZ:0-CUST-BILL-TO         PIC X(07).

```

Figure 236 COBOL Replacing Pseudo-Text, COBOL Copybook Member

Copied Member in VISION:Results

```

01 C-01-SLS04101-KEY.
05 C-01-0-SLS0R101-KEY.
   10 C-01-LOCN-ADMIN          PIC X(05).
   10 C-01-WORK-ORD-NUM        PIC 9(10).
05 C-01-CUST-NUM               PIC 9(07).
05 C-01-CUST-BILL-TO           PIC X(07).

```

Figure 237 COBOL Replacing Pseudo-Text, Copied Member in VISION:Results

Prefixing Data Names

The COPY feature allows for symbolic substitution as well as default values for these symbolics. You can take advantage of this feature when setting up VISION:Results copybooks describing files.

In the copybook, prefix each data name with a symbolic variable. #1CCENTER

Assign this variable a default value. #1=IN

In your program, when the COPY is requested with no symbolics specified, all data names are prefixed with IN. INCCENTER

You can specify a different prefix on the COPY command. #1=OUT

With this specified, all data names are prefixed with OUT. By setting up all VISION:Results copybooks in this manner, data names can be prefixed indicating the file and/or work area to which the fields belong.

```

COMPUTER ASSOCIATES    VISION:RESULTS

1-----VISION:RESULTS FREE FORM TEXT-----

FILE CCMSTR  FB  80  800
  COPY  CCENTER          <----
COPIED $DEFAULT  #1=IN
COPIED $DEND
COPIED      INCCENTER 4      (COST'CENTER)
COPIED      INOFFNO  4      (OFFICE'NUMBER)
COPIED      INEMPNO  6 (EMPLOYEE'NUMBER) INEMPNAME 19 (EMPLOYEE'NAME)
COPIED      INHDATE  6  NU  D  (HIRE'DATE)
COPIED      INSALCLASS 1 (SALARY'CLASS) INJCLASS 4 (JOB'CLASS)
COPIED      INLGRADE 1 (LABOR'GRADE) INSEX 1 (S'E'X) INMSTATUS 2
COPIED      INCRATE  6  NU  2  E (CURRENT'RATE)
COPIED      INIDATE  6  NU  D  (INCREASE'DATE) INITYPE 1 (INCREASE'TYPE)
COPIED      INIAMT  5  NU  2  E (INCREASE'AMOUNT)

FILE OCCMSTR FB  80  800
  COPY  CCENTER  #1=OUT      <----
COPIED $DEFAULT  #1=IN
COPIED $DEND
COPIED      OUTCENTER 4      (COST'CENTER)
COPIED      OUTOFFNO  4      (OFFICE'NUMBER)
COPIED      OUTEMPNO  6 (EMPLOYEE'NUMBER) OUTEMPNAME 19 (EMPLOYEE'NAME)
COPIED      OUTHDATE  6  NU  D  (HIRE'DATE)
COPIED      OUTSALCLASS 1 (SALARY'CLASS) OUTJCLASS 4 (JOB'CLASS)
COPIED      OUTLGRADE 1 (LABOR'GRADE) OUTSEX 1 (S'E'X) OUTMSTATUS 2
COPIED      OUTCRATE  6  NU  2  E (CURRENT'RATE)
COPIED      OUTIDATE  6  NU  D  (INCREASE'DATE) OUTITYPE 1 (INCREASE'TYPE)
COPIED      OUTIAMT  5  NU  2  E (INCREASE'AMOUNT)

WORKAREA
MESSAGE 20 AVG 6 NU 2 E (AVERAGE'RATE)
TALLY 4.0 (COUNT)

CONTROL INOFFNO INCCENTER
.
.
.

```

Figure 238 Prefixing Data Names

Using DB2 Tables

If you have a DB2 subsystem at your installation, you can copy field names from a DB2 table directly into your VISION:Results program. Use these generated field names for field definitions in FILE statements, work areas, or table areas. The syntax is as follows:

```
COPYDB2 tablename
```

Figure 239 COPYDB2 Command Syntax

You cannot have more than one DB2 table name for the COPYDB2 command, but you can have more than one COPYDB2 command.

Because storage is dynamically allocated to the number of fields of a DB2 table, COPYDB2 can retrieve an unlimited number of fields.

COPYDB2 Requirements

The DB2 subsystem ID and plan name are assigned through the VISION:Results installation macro (CATSYS and CATPLAN). If no information is provided by this macro, DB2A and DYLDB2 are used for subsystem ID and plan name, respectively.

If the subsystem ID and plan name differ from CATSYS and CATPLAN in the installation macro, you can override the default IDs using the following option:

```
OPTION CATPLANID xxxxxxxx CATSYSID nnnnnnnn
```

Figure 240 Override the Default IDs

where xxxxxxxx and nnnnnnnn are the subsystem and plan names, respectively.

If you do not want your program to be changed every time you use a different DB2 subsystem ID or plan, use the COPYDB2A DD statement method.

Note: If you want to use the COPYDB2 facility, you must have run either the DB2INSTL or DB2INST2 job stream in the source library at installation time. For more information, see the *Advantage VISION:Results for z/OS Installation Guide*.

COPYDB2A DD Statement to Override the Default IDs

Each COPYDB2 command can be dynamically assigned to its own subsystem ID and plan name without the need to modify the VISION:Results program directly. In addition, for each COPYDB2 command, the location and authorization IDs of the DB2 table can be dynamically added or updated.

Each COPYDB2 command is assigned a unique JCL DD statement that contains a single record containing the following positional parameters:

```
WORKAREA
SUBSYSID      8      1 ;DB2 subsystem ID
FILLER        1      9
PLANID        8     10 ;DB2 plan ID
FILLER        1     18
AUTHID        8     19 ;DB2 authorization ID
FILLER        1     27
LOCID       16     28 :DB2 location ID
```

The JCL overrides appear in the JCL as follows:

```
//COPYDB2A DD *
DB2T      DYLDB2   DSN8610
```

- The DB2 subsystem ID is in position 1 and can be up to eight characters.
- The DB2 plan name is in position 10 and can be up to eight characters.
- The table authorization ID qualifier is in position 19 and can be up to eight characters.
- The table location ID qualifier is in position 28 and can be up to 16 characters.

The first seven characters of each JCL DD statement are COPYDB2. The last character is unique for each COPYDB2 command and is assigned alphabetically in the order the COPYDB2 commands appear in the program. For example:

```
COPYDB2 DSN841.EMP      ==>    //COPYDB2A DD
COPYDB2 DSN841.DEPT    ==>    //COPYDB2B DD
COPYDB2 DSN841.PROJACCT ==>    //COPYDB2C DD
```

- The first COPYDB2 command in the program has its overrides in the input record assigned to the COPYDB2A DD statement;
 - The second COPYDB2 command in the program has its overrides in the input record assigned to the COPYDB2B DD statement;
 - The third COPYDB2 command in the program has its overrides in the input record assigned to the COPYDB2C DD statement;
- and so on.

If a COPYDB2A is not present in the JCL, there is no override for the first COPYDB2 command. If blanks are found where a positional parameter should be in the record, there is no override for that parameter. The same rules apply for other COPYDB2 commands and their corresponding COPYDB2 DD statements.

The CALL Attach Implicit Connection Facility

VISION:Results allows the COPYDB2 command to use the CALL Attach implicit connection. The CALL Attach implicit connection allows you to determine the DB2 subsystem ID to use independently of the source program. The DB2 subsystem ID is obtained from the DSNHDECP member of a DSNEXIT library, which must be concatenated to the program's STEPLIB. The DB2 plan name must be the same name as the COPYDB2 DBRM module, which is DYLCAT00.

Note: The COPYDB2A method is the standard way to override the DB2 data parameters independently of the source program. This method is another way to override the DB2 data parameters.

The CALL Attach implicit connection is implemented after running the VISION:Results DB2INST2 in the installation source library. The CATPLAN and CATSYS specified in the DYLINSTL macro are ignored, along with the CATPLANID and CATSYSID on the OPTION statement and the DB2 plan. In addition, the subsystem ID and plan name specified on the COPYDB2A DD statement are ignored.

The other parameters on the COPYDB2A DD statement are still functional. When the implicit connection is being used, the listings generate an IMPLICIT CONNECTION BEING USED comment line after the COPYDB2 command.

The standard CALL Attach explicit connection can still be used if an ECONNECT DD statement is present in the JCL.

TSO Attach Facility

VISION:Results allows COPYDB2 to be fully functional under the TSO Attach facility. The TSO Attach facility requires COPYDB2 to use the DB2 plan and subsystem ID specified within the SYSTSIN DD statement. This means it must use the same plan and subsystem ID that VISION:Results Interface to DB2 Dynamic uses.

The CATPLAN and CATSYS specified in the DYLINSTL macro are ignored, along with the CATPLANID and CATSYSID on the OPTION statement and the DB2 plan. In addition, the subsystem ID and plan name specified on the COPYDB2A DD statement are ignored.

The other parameters on the COPYDB2A DD statement are still functional. When the TSO Attach connection is being used, the listings generate a TSO ATTACH CONNECTION BEING USED comment line after the COPYDB2 command.

IMS Attach Facility

You can use the COPYDB2 command under the IMS Attach facility. You must use the JCL used for VISION:Results Interface to DB2. The IMS Attach capability works for both BMP and batch programs.

The COPYDB2 command uses the same IMS Attach facility that VISION:Results Interface to DB2 uses. For more information, see the VISION:Results Interface to DB2 documentation. The COPYDB2 command can use the IMS Attach facility only on a compile and go program. It cannot use the IMS Attach facility on a freeze.

The subsystem ID and plan name specified on the COPYDB2A DD statement are ignored. The other parameters on the COPYDB2A DD statement are still functional. When the IMS Attach facility is being used, the listings generate an IMS ATTACH CONNECTION BEING USED comment line after the COPYDB2 command.

VARCHAR Data Type Support

Previously, COPYDB2 retrieved a VARCHAR data type as a fixed field with the length being its maximum. For example, VISION:Results expanded the VARCHAR field as:

```
COPIED FIRSTNME 14 CH
COPIED REDEFINE AT FIRSTNME
COPIED FIRSTNME_LEN 2 BI
COPIED FIRSTNME_TEXT 12 CH
```

In r4.0 of VISION:Results Interface to DB2, VARCHAR data types are supported as both true variable fields and fixed fields. This means that a VARCHAR data type used as a variable field in LIST statements erroneously displays the binary length at the beginning of the field.

To fix this situation and remain compatible with existing programs that display and retrieve VARCHAR data type fields as fixed fields, the following COPYDB2 definition and expansion of VARCHAR data type fields are used:

```
COPIED FIRSTNME VAR 14 CH
COPIED REDEFINE AT FIRSTNME_VAR
COPIED FIRSTNME_LEN 2 BI
COPIED FIRSTNME_TEXT 12 CH
COPIED REDEFINE AT FIRSTNME_TEXT
COPIED FIRSTNME 12 CH
```

Generating Null Indicator Fields

COPYDB2 can include null indicator fields for all the fields that may have a null value. The null indicator field appears right after the field it reflects. It is a 2-byte binary field (required by DB2 SQL programming) and has the same name as the field it refers to, but with an additional ending suffix of _IND. If there is no room for the ending suffix, then the null indicator field is not built for that field.

Use DYLINSTL parameter DB2NULL to specify whether the null indicator fields are to be generated. If set to Y, the null indicator fields are generated. The default is to not use the DB2NULL parameter (DB2NULL=N).

COPYDB2 Example

The following example shows the VISION:Results listing using a COPYDB2 command.

```
FILE XFILE DUMMY
WORKAREA
  XXFNO 6 CH
  COPYDB2 DYLA.DYXEMPL
COPIED * EXPANDED COLUMN FIELDS OF TABLE DYLA.DYXEMPL
COPIED EMPNO 6 CH
COPIED FIRSTNME 12 CH
COPIED MIDINIT 1 CH
COPIED LASTNAME 15 CH
COPIED WORKDEPT 3 CH
COPIED PHONENO 4 CH
COPIED HIREDATE 10 CH ; DATE COLMN
COPIED JOBCODE 2 PD
COPIED EDUCVL 2 BI
COPIED SEX 1 CH
COPIED BRTHDATE 10 CH ; DATE COLMN
COPIED SALARY 5 PD 2
  WK1 8 NU
ON ONE
EXEC SQL VERSION
  DECLARE EMPSCR CURSOR FOR
  SELECT EMPNO, FIRSTNME, LASTNAME, SALARY, MIDINIT, WORKDEPT,
  HIREDATE, PHONENO, JOBCODE, EDUCVL, BRTHDATE, SEX
  FROM DYLA.DYXEMPL
  ORDER BY LASTNAME
ENDEXEC
ENDONE
:
```

Figure 241 COPYDB2 Command Example

JCL DB2 Overrides Example

```
//DYL280 EXEC RESULTS
//STEPLIB DD DISP=SHR,
//          DSN=YOUR.RESULTS.LOAD
//          DD DISP=SHR,
//          DSN=YOUR.DB2.LOAD
//          DD DISP=SHR,
//          DSN=YOUR.INTFDB2.LOAD
//COPYDB2A DD *
DB2T      DY28II4  DYL280
//COPYDB2B DD *
DB2T      DY28II3  SSI2  DB2B_WH
//SYSIN    DD *
OPTION CATSYSID DB2A CATPLANID DY280
FILE XFILE DUMMY
WORKAREA
COPYDB2 DY280
WORKAREA
COPYDB2 DSN8410.DEPT

STOP
/*
```

Remote DB2 server

```
COMPUTER ASSOCIATES          VISION:RESULTS 5.0 *
1-----VISION:RESULTS FREE FORM TEXT-----
OPTION CATSYSID DB2A CATPLANID DY280
FILE XFILE DUMMY
WORKAREA
COPYDB2 DY280
* OVERRIDE TABLENAME = DYL280.DY280
* OVERRIDE PLAN = DY28II4
* OVERRIDE SSID = DB2T
COPIED * EXPANDED COLUMN FIELDS OF TABLE DYL280.DY280
COPIED EMPNO                6 CH
COPIED FIRSTNAME VAR        14 CH
COPIED REDEFINE AT          FIRSTNAME_VAR
COPIED FIRSTNAME_LEN        2 BI
COPIED FIRSTNAME TEXT       12 CH
COPIED REDEFINE AT          FIRSTNAME_TEXT
COPIED FIRSTNAME            12 CH
COPIED MIDINIT              1 CH
COPIED LASTNAME VAR         17 CH
COPIED REDEFINE AT          LASTNAME_VAR
COPIED LASTNAME_LEN         2 BI
COPIED LASTNAME TEXT       15 CH
COPIED REDEFINE AT          LASTNAME_TEXT
COPIED LASTNAME             15 CH
COPIED WORKDEPT              3 CH
COPIED PHONENO              4 CH
COPIED HIREDATE              10 CH          ;DATE COLMN
COPIED JOB CODE              2 PD
COPIED EDUCATION            2 BI
COPIED SEX                   1 CH
COPIED BIRTHDATE            10 CH          ;DATE COLMN

COPIED SALARY                5 PD      2
WORKAREA
COPYDB2 DSN8410.DEPT
* OVERRIDE TABLENAME = DB2B_WH.SSI2.DEPT
* OVERRIDE PLAN = DY28II3
* OVERRIDE SSID = DB2T
COPIED * EXPANDED COLUMN FIELDS OF TABLE DB2B_WH.SSI2.DEPT
COPIED DEPTNO                3 CH
COPIED DEPTNAME VAR          38 CH
COPIED REDEFINE AT          DEPTNAME_VAR
COPIED DEPTNAME_LEN          2 BI
```

Figure 242 JCL DB2 Overrides Example (Page 1 of 2)

```
COPIED DEPTNAME TEXT          36 CH
COPIED REDEFINE AT DEPTNAME TEXT 36 CH
COPIED DEPTNAME              36 CH
COPIED MGRNO                  6 CH
COPIED ADMRDEPT               3 CH
COPIED LOCATION              16 CH

STOP
```

Figure 242 JCL DB2 Overrides Example (Page 2 of 2)

Chapter 16: Using Report Statements

When producing a report, you need to issue several commands to:

- Format the report.
- Establish the content of title and footing lines.
- Control the content and format of report detail lines and control break lines.
- Determine which fields are to cause control breaks.
- Alert VISION:Results as to what is to be done when a control break occurs.

The following statements deal specifically with producing a report in VISION:Results:

- REPORT (optional)
- LIST (required)
- CONTROL or SUBTOTAL (optional — required for ON CHANGE IN)
- ON CHANGE IN (optional)
- ON FINAL (optional)
- T1 to T9 (almost always required)

[Figure 243](#) shows a simple report and the VISION:Results statements used to produce the report.

Report Output

DEPARTMENT NET PAY SUMMARY STATEMENT			
DEPARTMENT	NAME	SOC. SEC. NO.	NET PAY
1	CARLSON	345-66-2113	645.78
	JAMES	893-24-5673	786.99
**DEPARTMENT TOTAL			1432.77
2	MEYERS	678-54-3456	854.34
	ROGERS	213-99-7987	775.44
**DEPARTMENT TOTAL			1629.78
***AVERAGE NET PAY			765.66

Report Statement

TITLE statement (T1-T9)
Field definition or [LIST Statement](#)
[LIST Statement](#)

CONTROL LIST

[ON CHANGE IN](#) statements

[ON FINAL](#) and TALLY statements

Figure 243 Simple Report and VISION:Results Statements

[Report Statement](#) describes each of the above statements. Examples are provided at the end of this chapter in [Report Examples on page 412](#).

Report Statement

The REPORT statement is optional and need only be provided if you want to override the report writer defaults and exit from report logic to a user-written external subroutine.

The REPORT statement must precede any LIST or Tn (TITLE) statements and, in most cases, should be the first statement in your VISION:Results program.

The format of the REPORT statement is described in the following sections.

z/OS

```
REPORT [nnn] [nnn WIDE] [DELIM [C'c' | X'xx']] [nnn LONG] [nn BETWEEN]  
[ASA|DYLPRCx] [SYS280Rx] [HTML ddname [COUNT nnnn] [STYLE nn] [HTMLTITL dataname] ]
```

Figure 244 REPORT Statement Format—z/OS

For z/OS only, if SYSBLOK was selected in the DYLINSTL macro and no JCL block size was specified on the SYS280Rx DD statement, then the operating system calculates the block size. If SYSBLOK is not used, the block size is set to 8 times the record length.

VSE

For users of VSE r2.1 or later, a tape or disk device can be specified in the REPORT statement using the PLUnnn option, as well as optionally defining a blocking factor. This allows direct specification of a disk or tape unit, rather than a printer device, as the destination of the report produced.

```
REPORT [nnn] [nnn WIDE] [DELIM[C'c'|X'xx']] [nnn LONG] [ nn BETWEEN]  
[DYL280Ra | [ [ASA] [PLUnnn] [DISK [nn] |  
[TAPE [nn] [NL|SL|T] [REWIND|NORWD|UNLOAD]]]]]
```

Figure 245 REPORT Statement Using the PLUnnn Option - VSE

For example,

```
REPORT 80 WIDE PLU200 DISK 50
```

specifies PLU200 as the name on a DLBL statement for a disk file to contain the report. The file is blocked 50 (81x50) when written (the default is 10).

Note: If the PLU on the report statement is not referring to a disk or tape, then you must use SYSnnn in the JCL. Otherwise, use PLUnnn in the JCL.

Set the nn following TAPE or DISK for PLUnnn to the desired blocking factor for the report file (the default is 10). For tape files, provide tape label and rewind information with the information in [TAPE Label Rewind \(Options\) on page 121](#).

nnn WIDE

The keyword WIDE defines the number of characters-per-line wanted. The number (nnn) must precede the keyword. The default in VISION:Results is 132 characters per line. VISION:Results permits from 1 to 204 characters per line.

```
REPORT 120 WIDE
REPORT 80 WIDE
REPORT 204 WIDE
REPORT 145 WIDE
```

DELIM

DELIM instructs the report writer logic to:

- Format the print lines with no carriage control, titles, or column headings.
- Format the print lines with one space between columns, which is filled with the DELIM-specified character. However, if fixed print positions are used, then columns will be formatted based on the specified positions. The use of fixed print positions requires you to adjust the print positions to prevent truncation of fields. If DELIM is coded on the REPORT statement without a specified character, the value supplied in the DYLINSTL macro is used. An error is generated if no DYLINSTL value has been provided. You must select a character for the DELIM-specified character that does not occur in the data to be sent to the PC. You can use any 1-character value except X'40 or C' '(blank) as the DELIM value. The delimiter does not occur at the beginning of the record unless the DYLINSTL DLMFRST parameter is set to Y. If DLMFRST is set to Y, then the specified character occurs in file position 2, because the first position is always reserved for carriage control.
- Generate the output to a data set rather than a printer.

The data set can then be downloaded to a PC.

```
REPORT 100 WIDE DELIM C'$'
FILE INFILE F 80 CARDS
ACCOUNT 4 CH
NAME 18 CH
BALANCE 9 NU 2
LIST ACCOUNT NAME BALANCE X
FIN
1023 GEORGE JONES000123499
1034ALEXANDER HAMILTON00004002K
```

Figure 246 Sample DELIM Program

This would create the following data set on SYS280RA if DLMFRST is set to Y:

```
$1023$ GEORGE JONES$ 1,234.99$
$1034$ALEXANDER HAMILTON$ -400.22$
```

Figure 247 Sample DELIM Program Output

VISION:Results produces a warning message stating that the LONG keyword is ignored if both the LONG keyword and the DELIM keyword are present on the REPORT statement.

You cannot use LIST statements with SUPPRESS and DELIM; VISION:Results produces an error message.

nnn LONG

The keyword LONG overrides the number of lines per page. The number (n) specifying lines per page includes title lines, column heading lines, report body lines, and footings. Blank lines are also counted. The number (n) must precede the keyword LONG. VISION:Results permits from 1 to 999 lines per page. The default in VISION:Results is 55 lines per page, but it can be changed using the DYLINSTL parameter PGLINES. If the DELIM keyword is specified on the REPORT statement, the value of LONG is ignored. For details, see the *Advantage VISION:Results for z/OS Installation Guide*.

```
REPORT 54 LONG
REPORT 60 LONG
REPORT 54 LONG 120 WIDE
```

VISION:Results produces a warning message stating that the LONG keyword is ignored if both the LONG keyword and the DELIM keyword are present on the REPORT statement.

nn BETWEEN

The maximum number of spaces between columns can be overridden using the keyword BETWEEN. VISION:Results permits from 1 to 99 positions between columns. The default number of positions between columns is 5.

When the DYLINSTL EDALIGN parameter is set to Y, all the edit codes use approximately the same right-justified alignment for numeric fields. If the default positioning for the BETWEEN parameter is used, the numeric fields are adjusted to fit between the previous and subsequent columns. However, if the BETWEEN parameter has a value other than 5, the edit code alignments for numeric fields use the same alignment that the default edit code (P edit code format) uses when EDALIGN is set to N. The value of the BETWEEN parameter should be adjusted according to the size of the numeric field to prevent overlapping or truncation between other column fields on the report line.

Note: The EDALIGN DYLINSTL parameter only affects numeric fields specified on the first LIST command. It does not affect character fields defined either explicitly, or implicitly output by the EDIT command.

If the DELIM keyword is specified on the REPORT statement, any value specified on BETWEEN is ignored, and a value of 1 is substituted.

```
REPORT 10 BETWEEN
REPORT 3 BETWEEN
REPORT 2 BETWEEN 60 LONG 128 WIDE
REPORT 204 WIDE 50 LONG 10 BETWEEN
```

For more information, see [SYS280Rx on page 454](#).

Modulename | ASA

The module name and ASA are mutually exclusive.

Module Name

The module name is the phase (VSE) or member (z/OS) name of your subroutine that interrogates each print line of the report. The subroutine must have been previously link edited to a Load (z/OS) or Core Image Library (VSE). The subroutine module is loaded into memory prior to execution and is entered before each print line is written.

The module name must comply with the following restrictions:

z/OS	The module name of your external subroutine must be named DYLP RCx (where x is an alpha character from B through Z).
VSE	The module name of your external subroutine must be named DY L280 Rx (where x is an alpha character from B through F).

For additional information, see [Report Print Line Exit—Your Exit Routines on page 427](#).

ASA

The keyword ASA provides you with ASA carriage control characters instead of standard machine control characters. If ASA is used, the resulting report can be written to a printer or any other device. ASA carriage control should be used only when using equipment that does not recognize machine carriage control (such as microfiche, microfilm, or offline printers).

SYS280Rx

SYS280Rx indicates to which DD the report is to be written. The x is any character valid in a ddname, including a blank. If you are producing multiple reports, you may want some of the reports written to a different DD. VISION:Results writes the report to the ddname specified on your REPORT statement.

For HTML processing, the SYS280Rx ddname is used for debugging purposes. It reflects what the HTML reports will look like prior to the HTML formatting process.

HTML ddname

The JCL ddname following the HTML keyword on the REPORT statement relates to either a z/OS PDS or an HFS PATH name.

You can use the REPORT and LIST statements to create an HTML document that can be viewed by a Web browser. VISION:Results includes an HTML skeleton containing the prologue text, which is copied to the output report, followed by the report lines.

You must add a new JCL DD statement to the execution JCL, with the DDNAME of VRHTMLIB, which points to the provided HTML library. You also need an additional DDNAME (VRHTMWRK) for a work file to be used during the creation of the HTML report.

Do not use carriage control statements such as WITH nn | EJECT AFTER in the report definition statements. The generated report lines do not contain any carriage control characters. The output data set, identified by the HTML keyword on the REPORT statement, can point to a z/OS PDS or an HFS path.

- HTML document member HEAD contains a single copy of the Title lines and column headings, so do not modify titles during program execution.
- HTML document member BDYnnnnn contains the report lines. When using HTML, the keyword LONG default value is 25.
- HTML document member LTOC contains a table of contents frame using the changed value of a field specified in a LIST SUPPRESS statement.

Only the last suppressed field appearing on a single LIST SUPPRESS statement is used for the LTOC frame, so you should use SUPPRESS only for a single field in your LIST statement. Each item in the LTOC links to its corresponding data in the body of the report.

- If the report contains a large number of lines, you can break the body of the report into multiple segments in the output file. This allows the browser to retrieve the report lines more rapidly.
- The number of lines per report segment is specified on the REPORT statement, after the ddname of the output document:

```
REPORT ... [HTML ddname [COUNT nnnn] [STYLE nn]]
```

where ddname is the JCL DDNAME of the PDS or HFS path name that will contain the HTML document members. The default COUNT is 500 lines. The maximum value is 9999.

The values of nn for STYLE are 00 (default) through 99. Use the STYLE number to select the HTML style member from the supplied VRHTMLIB partitioned data set. The style number is used as a suffix to the member name. Two styles, or sets of members, are provided: suffix 00 (the default) and suffix 01. You can add members with suffix values from 02-99 with customized colors or fonts for the HTML document.

HTMLTITL Dataname

The HTMLTITL dataname allows the title and column headings to remain statically positioned at the top of the document. It also allows the title frame, header frame, and body frame to scroll together horizontally, thus keeping column headings positioned above the report text.

The HTMLTITL dataname lets you to specify a line of text that will be displayed on the top (information) line of the browser display.

Note: This is *not* a title that will be displayed in the actual report. It is the title of the web browser page.

HTMLTITL Syntax

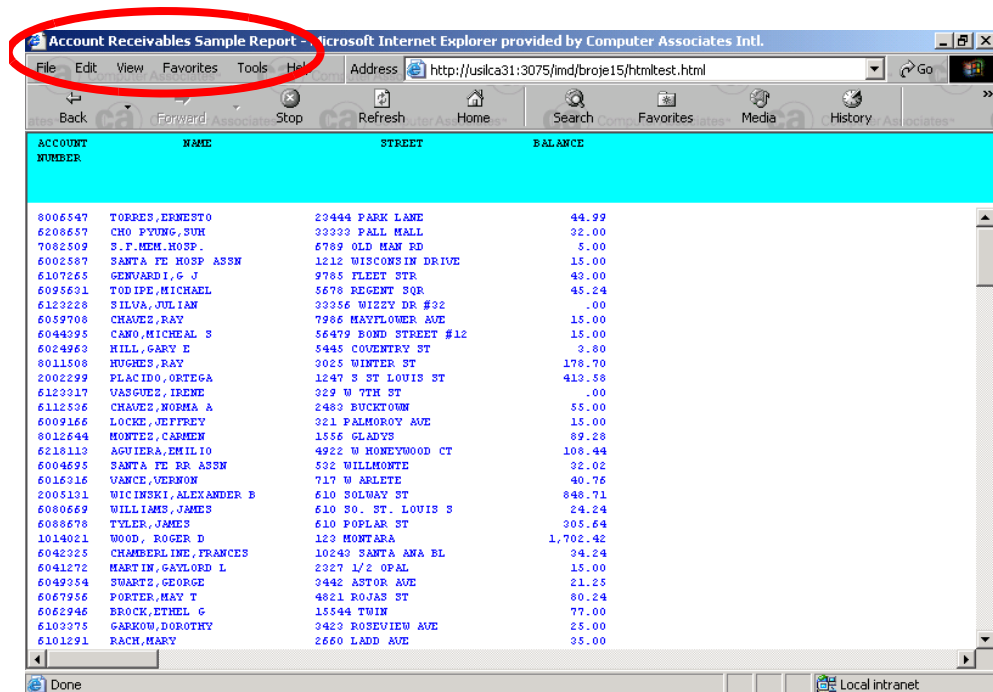
```
REPORT ... [HTML HTMLOUT STYLE nn HTMLTITL dataname]
```

where dataname is the name of a character field that will contain the text to be displayed on the information line of the browser page when viewing the HTML produced by the REPORT statement. The maximum length of the text to be displayed is 80 characters. The maximum length of the dataname is 10 characters.

For example, if the following was coded in the program:

```
WORKAREA
  WEBTITLE 40 CH VALUE
  "Account Receivables Sample Report"
REPORT 80 WIDE HTML HTMLOUT STYLE 00 HTMLTITL WEBTITLE
```

The following title would be displayed on the web browser's tool bar:



Member Names

The member names in the provided template library are:

Member Name	Description
HTBODYxx	Defines the structure of the body frame for the document.
HTLTOCxx	Defines the Table of Contents frame for the document, which will be created if LIST SUPPRESS is used in the report, and will be displayed in the left margin of the document. When browsing the HTML document, select an entry in the Table of Contents frame to scroll the body text frame to the corresponding location.
HTHEADxx	Defines the color and font used to create the document.
HTPRIXxx and HTPRIYxx	Define the HTML frames to be used when creating the document. HTPRIXxx is used to produce a document with a Table of Contents frame. HTPRIYxx is used to create a document without the Table of Contents frame.

The generated HTML PDS or HFS path contains the following member names or HFS files:

Member Name	Description
\$MAIN	Defines the HTML frames for the document.
HEAD	Contains the heading frame for the report.
LTOC	Contains the hyperlinks that can be used to navigate through a long report.
BDYnnnnn	Contains one or more body frames. Multiple members are created when the size of a body text member exceeds the COUNT value.

Naming Conventions

When you create an HTML document as an HFS file, the following naming conventions are used:

- If a file name is not provided on the DD statement, VISION:Results names the HFS file **membername.html**.
- If a file name is provided, VISION:Results names the HFS file **yourfilename-membername.html**, where **yourfilename** is the file name that you use.
- If a file name contains an HTML extension, the \$MAIN file is named **yourfilename.html**.
- If a file name is provided without an HTML extension, the \$MAIN file is named **yourfilename**.

Using HTML

VISION:Results supports HTML documents that can be accessed from a z/OS product (such as IBM's WebSphere Application Server for z/OS, hereafter referred to as Websphere). HTML documents are created using the HTML template library provided on the VISION:Results installation tape. Once the HTML documents are created, you can use WebSphere to browse the PDS containing the document, or transfer the PDS or HFS files to a server or local workstation for viewing on a web browser.

The HTML produced for a PDS is specifically designed to work with WebSphere on a z/OS host system. If WebSphere is not available, the HTML link statements (such as src= and href=) within the PDS member must be modified to conform with the required syntax for the platform where the HTML documents will eventually reside. This is necessary only for a PDS file; HFS files can be transferred without any editing.

If WebSphere is not available, the members must be transferred using a suitable file transfer tool to a server or local workstation where they can be accessed by the web browser. The browser session used to view the VISION:Results HTML document must initiate the viewing using the \$MAIN file (member).

To use this feature, LE must be enabled either through the OPTION statement or DYLINSTL LE=Y.

Creating HTML Documents Using HFS Path

[Figure 248](#) is an example of JCL to create an HTML document using an HFS path:

```
//DYL1 EXEC PGM=DYL280,REGION=0M
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(4,4))
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//OUTPUT DD SYSOUT=*
//***** HTML LIBRARY *****
//VRHTLIB DD DSN=YOUR.HTML.LIBRARY,DISP=SHR
//***** HTML WORK FILE *****
//VRHTWRK DD UNIT=SYSDA,SPACE=(TRK,(10,5))
//SYS280R DD SYSOUT=*
//*****
//HTMLOUT DD PATH='/your/html/path/name...',
//          PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//          PATHMODE=(SIRUSR,SIWUSR)
//AROUT DD DSN=your.arout.file,DISP=(,CATLG),
//          UNIT=SYSDA,SPACE=(TRK,(2,1)),
//          DCB=(RECFM=FB,LRECL=352,BLKSIZE=5280)
//SYSIN DD *
```

Figure 248 JCL to Create an HTML Data Set Using HFS Path

[Figure 249](#) shows the VISION:Results statements:

```
FILE ARFILE FB 352 5280 STATUS ARSTAT
REPORT 80 WIDE HTML HTMLOUT STYLE 00
.
.
.
```

Figure 249 VISION:Results Statements to Create an HTML Data Set Using HFS Path

The HFS path name in the above JCL example must be enclosed with single quotation marks.

[Figure 249](#) illustrates how to use the Report statement with the HTML ddname and STYLE keyword. It is not a complete VISION:Results program.

[Figure 250](#) is the \$MAIN member produced from the JCL and VISION:Results statements in [Figure 248](#) and [Figure 249](#).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<TITLE>VISION:Results Report Type X, Default Style</TITLE>
</HEAD>
<FRAMESET COLS="19%,*">
  <FRAME NAME="ltoc" SRC="mytest-ltoc.html" MARGINHEIGHT=0 MARGINWIDTH=0>
  <FRAMESET ROWS="13%,*" BORDER=1>
    <FRAME NAME="head" SRC="mytest-head.html" MARGINHEIGHT=2 MARGINWIDTH=2
      SCROLLING=NO>
    <FRAME NAME="body" SRC="mytest-bdy00001.html" MARGINHEIGHT=2 MARGINWIDTH=2>
  </FRAMESET>
</FRAMESET>
</HTML>
```

Figure 250 HFS \$MAIN Member with Default Style 00

[Figure 251](#) is the HEAD member produced from the JCL and VISION:Results statements in [Figure 248](#) and [Figure 249](#).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<TITLE>VISION:Results Report Heading Frame, Default Style</TITLE>
<STYLE>
<!--
BODY {color:black; background:aqua}
PRE {color:black; background:aqua; font-size:x-small}
-->
</STYLE>
</HEAD>
<BODY>
<PRE>
06/26/2001                THIS IS THE HEAD MEMBER                PAGE        1
<PRE>
      ACCOUNT      ACCOUNT      NAME
      CODE        NUMBER
</PRE>
</BODY>
</HTML>
```

Figure 251 HFS HEAD Member with Default Style 00

[Figure 252](#) is the LTOC member produced from the JCL and VISION:Results statements in [Figure 248](#) and [Figure 249](#).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<BASEFONT SIZE=2>
<HEAD>
<TITLE>VISION:Results Report Table of Contents Frame,
      Default Style</TITLE>
<STYLE>
<!--
BODY {color:black; background:skyblue}
H4   {color:yellow; background:blue; font-size:medium; font-weight:bold}
A    {color:maroon; background:skyblue; font-size:x-small}
-->
</STYLE>
</HEAD>
<BODY>
<H4>Report Contents</H4>
<UL>
<LI><A HREF="mytest-bdy00001.html#ref00001" TARGET="body">BO</A>
<LI><A HREF="mytest-bdy00001.html#ref00002" TARGET="body">EO</A>
</UL>
</BODY>
</HTML>
```

Figure 252 HFS LTOC Member with Default Style 00

[Figure 253](#) is the BDY00001 member produced from the JCL and VISION:Results statements in [Figure 248](#) and [Figure 249](#).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<TITLE>VISION:Results Report Detail Frame, Default Style</TITLE>
<STYLE>
<!--
BODY {color:blue; background:white}
H5   {color:black; background:lightgrey; font-size:x-small;
      font-weight:bold; font-family:courier;
      text-align:left; line-height:normal}
PRE  {color:blue; background:white; font-size:x-small}
-->
</STYLE>
</HEAD>
<BODY>
<PRE>
A NAME="ref00001"></A><H5>BO</H5>
      BO              8006547          TORRES, ERNESTO          44.99

A NAME="ref00002"></A><H5>EO</H5>
      EO              6208657          CHO PYUNG, SUH           32.00
```

Figure 253 HFS BDY00001 Member with Default Style 00

[Figure 254](#) is an example of an HFS path directory on the z/OS host using OpenMVS ISPF Shell.

```
/your/hfs/path/name/
Select one or more files with / or action codes.

Type   Changed (GMT)   Owner   Filename
File   07/30/2001 20:51   USER   mytest
File   07/30/2001 20:51   USER   mytest-bdy00001.html
File   07/30/2001 20:51   USER   mytest-head.html
File   07/30/2001 20:51   USER   mytest-ltoc.html
```

Figure 254 HTML Member Names in the HFS Path Directory

Creating HTML Documents Using HTML PDS

[Figure 255](#) is an example of JCL to create an HTML PDS.

```
//DYL1 EXEC PGM=DYL280,REGION=0M
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(4,4))
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//OUTPUT DD SYSOUT=*
//***** HTML LIBRARY
//VRHTMLIB DD DSN=YOUR.HTML.LIBRARY,DISP=SHR
//***** HTML WORK FILE
//VRHTMWRK DD UNIT=SYSDA,SPACE=(TRK,(10,5))
//SYS280R DD SYSOUT=*
//*****
//HTMLOUTP DD DSN=YOUR.HTML.PDS,DISP=(,CATLG),
// UNIT=SYSDA,SPACE=(TRK,(5,2,5))
//AROUT DD DSN=your.arout.file,DISP=(,CATLG),
// UNIT=SYSDA,SPACE=(TRK,(5,2,5)),
// DCB=(RECFM=FB,LRECL=352,BLKSIZE=5280)
//SYSIN DD *
```

Figure 255 JCL to Create an HTML PDS

In [Figure 255](#), the DCB parameters for allocating YOUR.HTML.PDS are omitted because VISION:Results automatically supplies the required DCB information as a variable block record format with a record length of 1008.

[Figure 256](#) shows the VISION:Results statements.

```
FILE ARFILE FB 352 5280 STATUS ARSTAT
REPORT 80 WIDE HTML HTMLOUTP STYLE 00
.
.
.
```

Figure 256 VISION:Results Statements to Create an HTML PDS

[Figure 256](#) illustrates how to use the Report statement with the HTML ddname and STYLE keyword. It is not a complete VISION:Results program.

[Figure 257](#) is the \$MAIN member in the HTML PDS produced from the JCL and VISION:Results statements in [Figure 255](#) and [Figure 256](#).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<TITLE>VISION:Results Report Type X, Default Style</TITLE>
</HEAD>
<FRAMESET COLS="19%,*">
  <FRAME NAME="ltoc" SRC="/MVSDB/YOUR.HTMLOUT.DATASET(LTOC)'" MARGINHEIGHT=0 MARGINWIDTH=0>
  <FRAMESET ROWS="13%,*" BORDER=1>
    <FRAME NAME="head" SRC="/MVSDB/YOUR.HTMLOUT.DATASET(HEAD)'" MARGINHEIGHT=2 MARGINWIDTH=2>
      SCROLLING=NO
    <FRAME NAME="body" SRC="/MVSDB/YOUR.HTMLOUT.DATASET(BDY00001)'" MARGINHEIGHT=2 MARGINWIDTH=2>
  </FRAMESET>
</FRAMESET>
</HTML>
```

Figure 257 \$MAIN Member with Default Style 00

[Figure 258](#) is the HEAD member in the HTML PDS produced from the JCL and VISION:Results statements in [Figure 255](#) and [Figure 256](#).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<TITLE>VISION:Results Report Heading Frame, Default Style</TITLE>
<STYLE>
<!--
BODY {color:black; background:aqua}
PRE {color:black; background:aqua; font-size:x-small}
-->
</STYLE>
</HEAD>
<BODY>
<PRE>
    06/04/2001          THIS IS THE HEAD PDS MEMBER          PAGE      1
      ACCOUNT          ACCOUNT          NAME          BALANCE
      CODE            NUMBER
</PRE>
</BODY>
</HTML>
```

Figure 258 HEAD Member with Default Style 00

[Figure 259](#) is the LTOC member in the HTML PDS produced from the JCL and VISION:Results statements in [Figure 255](#) and [Figure 256](#).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<BASEFONT SIZE=2>
<HEAD>
<TITLE>VISION:Results Report Table of Contents Frame,
      Default Style</TITLE>
<STYLE>
<!--
BODY {color:black; background:skyblue}
H4 {color:yellow; background:blue; font-size:medium; font-weight:bold}
A {color:maroon; background:skyblue; font-size:x-small}
-->
</STYLE>
</HEAD>
<BODY>
<H4>Report Contents</H4>
<UL>
<LI><A HREF="/MWSDS/'YOUR.HTMLOUT.DATASET(BDY00001)'#ref00001"TARGET="body">BO</A>
<LI><A HREF="/MWSDS/'YOUR.HTMLOUT.DATASET(BDY00001)'#ref00001"TARGET="body">BO</A>
</UL>
</BODY>
</HTML>
```

Figure 259 LTOC Member with Default Style 00

[Figure 260](#) is the BDY00001 member in the HTML PDS produced from the JCL and VISION:Results statements in [Figure 255](#) and [Figure 256](#).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<TITLE>VISION:Results Report Detail Frame, Default Style</TITLE>
<STYLE>
<!--
BODY {color:blue; background:white}
H5 {color:black; background:lightgrey; font-size:x-small;
    font-weight:bold; font-family:courier;
    text-align:left; line-height:normal}
PRE {color:blue; background:white; font-size:x-small}
-->
</STYLE>
</HEAD>
<BODY>
```

Figure 260 BDY00001 member with Default Style 00 (Page 1 of 2)

```

<PRE>
A NAME="ref00001"></A><H5>BO</H5>
      BO      8006547      TORRES, ERNESTO      44.99
                                         44.99
A NAME="ref00002"></A><H5>EO</H5>
      EO      6208657      CHO PYUNG, SUH      32.00
                                         32.00
</BODY>
</HTML>

```

Figure 260 BDY00001 member with Default Style 00 (Page 2 of 2)

Dynamic Allocation of HTML Using HFS Path

[Figure 261](#) and [Figure 262](#) are examples of JCL and VISION:Results statements to dynamically allocate HTML files using an HFS path. The VISION:Results program includes the JCL DD statements to do this.

```

//DYL1 EXEC PGM=DYL280,REGION=0M
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(4,4))
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//OUTPUT DD SYSOUT=*
//***** SORT
//SYSOUT DD SYSOUT=*
//SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(15),,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(15),,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(15),,CONTIG)
//SORTWK04 DD UNIT=SYSDA,SPACE=(CYL,(15),,CONTIG)
//SORTWK05 DD UNIT=SYSDA,SPACE=(CYL,(15),,CONTIG)
//***** HTML LIBRARY
//VRHTMLIB DD DSN=YOUR.HTML.LIBRARY,DISP=SHR
//***** HTML WORK FILE
//VRHTMWRK DD UNIT=SYSDA,SPACE=(TRK,(10,5))
//SYS280R DD SYSOUT=*
//SYSIN DD *

```

Figure 261 JCL to Dynamically Allocate Files Using HFS Path

[Figure 262](#) shows the VISION:Results statements, but contains only those VISION:Results statements required to illustrate the use of the ALLOCATE facility for HTML. It is not a complete VISION:Results program.

```

REPORT HIML HIMLOUT STYLE 00
.
.
*****
* DYNAMIC ALLOCATION JCL DD STATEMENTS ARE BELOW.
*****

WORKAREA
JCLSTAT 1
MSG1 41 VALUE 'SYNTAX ERROR IN DYNAMICALLY ALLOCATED'
      'JCL'
MSG2 47 VALUE 'ALLOCATION FAILURE IN DYNAMICALLY'
      'ALLOCATED JCL'

WORKAREA
OUTJCL      240 1
FILLER      80 1 VALUE
  "/HIMLOUT DD PATH='/your/hfs/path/name...',"
FILLER      80 1 VALUE
  "// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),"
FILLER      80 1 VALUE
  "// PATHMODE=(SIRUSR,SINUSR)"

```

Figure 262 VISION:Results Statements to Dynamically Allocate Files Using an HFS Path (Page 1 of 2)

```

ON ONE
ALLOCATE OUTJCL STATUSFLAG JCLSTAT
  IF JCLSTAT EQ 'N'
    PRINT MSG1
  STOP
ENDIF
  IF JCLSTAT EQ 'F'
    PRINT MSG2
  STOP
ENDIF
ENDONE

*****
* END OF DYNAMIC ALLOCATION JCL DD STATEMENTS.
*****

.
* move data to the outrec
WRITE ABOUT

```

Figure 262 VISION:Results Statements to Dynamically Allocate Files Using an HFS Path (Page 2 of 2)

To dynamically allocate HTML files using an HFS path, you must enclose each value statement with double quotations, as shown in [Figure 262](#). The HFS path name must be enclosed in single quotations.

If the HFS path name extends to two or more JCL lines, then the JCL required for Dynamic Allocation should be read and moved into working storage, as demonstrated in [Example 2—Allocate Facility and MOVE Command in OUTJCL Fields on page 321](#).

Dynamic Allocation of an HTML Partitioned Data Set

[Figure 263](#) and [Figure 264](#) are examples of JCL and VISION:Results statements to dynamically allocate an HTML PDS. The VISION:Results program includes the JCL DD statements to do this.

```

//DYL1 EXEC PGM=DYL280,REGION=0M
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(4,4))
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//SYSABEND DD SYSOUT=*
//OUTPUT DD SYSOUT=*
//***** SORT
//SYSOUT DD SYSOUT=*
//SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(15),,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(15),,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(15),,CONTIG)
//SORTWK04 DD UNIT=SYSDA,SPACE=(CYL,(15),,CONTIG)
//SORTWK05 DD UNIT=SYSDA,SPACE=(CYL,(15),,CONTIG)
//***** HTML LIBRARY
//VRHTMLIB DD DSN=YOUR.HTML.LIBRARY,DISP=SHR
//***** HTML WORK FILE
//VRHTMWRK DD UNIT=SYSDA,SPACE=(TRK,(10,5))
//SYS280R DD SYSOUT=*
//SYSIN DD *

```

Figure 263 JCL to Dynamically Allocate an HTML PDS

[Figure 264](#) shows the VISION:Results statements, but contains only those statements required to illustrate the use of the ALLOCATE facility for HTML. It is not a complete VISION:Results program.

```
REPORT  HML HMLOUTP STYLE 00
.
.
*****
* DYNAMIC ALLOCATION JCL DD STATEMENTS ARE BELOW.
*****

WORKAREA
  JCLSTAT 1
    MSG1 41 VALUE 'SYNTAX ERROR IN DYNAMICALLY ALLOCATED'
      'JCL'
    MSG2 47 VALUE 'ALLOCATION FAILURE IN DYNAMICALLY'
      'ALLOCATED JCL'

WORKAREA
  OUTJCL          160  1
  FILLER          80  1 VALUE
  '//HMLOUTP DD PATH=YOUR.HTML.PDS,DISP=(,CATLG),'
  FILLER          80  VALUE
  '//          UNIT=SYSDA,SPACE=(TRK,10,5,2))'

ON ONE

ALLOCATE OUTJCL STATUSFLAG JCLSTAT
  IF JCLSTAT EQ 'N'
    PRINT MSG1
    STOP
  ENDIF
  IF JCLSTAT EQ 'F'
    PRINT MSG2
    STOP
  ENDIF
ENDONE

*****
* END OF DYNAMIC ALLOCATION JCL DD STATEMENTS.
*****
.
* move data to the outrec

WRITE ABOUT
```

Figure 264 VISION:Results Statements to Dynamically Allocate an HTML PDS

The DCB parameters are not required in [Figure 264](#) to dynamically allocate the HTML PDS. VISION:Results automatically allocates the HTML PDS with a variable block record format that has a record length of 1008.

HTML Document on the Web Browser PC

Figure 265 is the final HTML document displayed on your web browser. It includes the \$MAIN, HEAD, LTOC, and BDYnnnnn members that define the entire report.

Report Contents

06/26/2001 THIS IS THE HEAD MEMBER PAGE 1
CREATE HTML DOCUMENT

ACCOUNT CODE	ACCOUNT NUMBER	NAME	BALANCE
BO	8006547	TORRES, ERNESTO	
EO	6208657	CHO PYUNG, SUH	44.99
	7082509	S.F.MEM.HOSP.	5.00
	6002587	SANTA FE HOSP ASSN	15.00
			52.00
FO	6107265	GENVARDI, G J	
	6095631	TODIPE, MICHAEL	45.24
	6123228	SILVA, JULIAN	.00
	6059708	CHAVEZ, RAY	15.00
	6044395	CANO, MICHAEL S	15.00
	6024963	HILL, GARY E	3.80
	8011508	HUGHES, RAY	178.70
			300.74
IO	2002299	PLACIDO, ORTEGA	

Figure 265 HTML Document That Displays on Web Browser

PLUnnn

The keyword PLUnnn (Programmer Logical Unit) is for VSE users only. If used, the nnn must contain a valid 3-digit SYS number. If PLUnnn is specified in the REPORT statement, the report is written to this programmer logical unit rather than to SYSLST. The report can be written to disk or tape or to another printer. This is useful when you want to separate the program or file listing from the report itself.

If you have IBM's POWER spooling system on your system, you can assign your report to a logical printer, as follows:

```
// ASSGN  SYSnnn,x'cuu'
* $$ LST  LST=cuu,FNO=xxxx
```

The keyword PLUnnn can be used with ASA in the REPORT statement. However, the PLUnnn cannot be used with any module name; they are mutually exclusive.

```
REPORT PLUnnn
REPORT PLUnnn ASA
REPORT ASA 120 WIDE 50 LONG 10 BETWEEN PLUnnn
```

Figure 266 PLUnnn Keyword Used with ASA

VSE users at r2.1 or later can specify a disk or tape drive on the REPORT statement using the keyword PLUnnn. This allows direct specification of a disk or tape, rather than a printer driver, as a destination of the report produced.

```
REPORT 80 WIDE PLU200 DISK 50
```

This specifies PLU200 as the name on a DLBL statement for a disk file to contain the report. The file will be blocked 50 (81 x 50) when written (the default is 10).

```
REPORT 100 WIDE PLU200 TAPE SL UNLOAD
```

This specifies PLU200 as the PLU (Programmer Logical Unit) number on an ASSGN statement for a tape file to contain the report. The file will be written as 101-byte records, blocked 10, on a standard labeled tape, which is rewound and unloaded at end of job.

AND

The keyword AND improves the readability of the REPORT statement. AND is a noise word and is never required.

```
REPORT ASA 120 WIDE AND 50 LONG AND 10 BETWEEN  
REPORT 50 LONG AND 130 WIDE
```

Because all of the information in the REPORT statement is optional, you can supply the REPORT statement in your VISION:Results program with no overrides:

```
REPORT
```

The following examples show how to override various defaults. To override the defaults for characters-per-line and lines-per-page:

```
REPORT 100 WIDE 60 LONG
```

or

```
REPORT 100 WIDE AND 60 LONG
```

or

```
REPORT 60 LONG 100 WIDE
```

or

```
REPORT 60 LONG AND 100 WIDE
```

To override the maximum number of spaces between columns (all other defaults are accepted):

```
REPORT 10 BETWEEN
```

To override all defaults:

```
REPORT 100 WIDE AND 60 LONG 10 BETWEEN
```

or

```
REPORT 100 WIDE 60 LONG 10 BETWEEN
```

or

```
REPORT 100 WIDE AND 60 LONG AND 10 BETWEEN
```

To exit to a user-written external subroutine:

```
VSE:      REPORT DYL280RB
OS/390:   REPORT DYLPSCP 204 WIDE
```

Title and Footing Statements

The TITLE or FOOTING statement defines both the contents and placement of the title or footing lines on your report. You can specify up to a maximum of nine title and/or footing lines per report. If specified in the VISION:Results program, these title or footing lines are printed automatically on each page of the report. The format of the TITLE/FOOTING statement is:

```
Tn 'title contents' [FIXED] [FOOTING] [WITH n AFTER]
```

Figure 267 TITLE or FOOTING Statement

Tn

Defining report headings or footings requires a T1 through T9 statement followed by the title contents in single quotation marks. A maximum of nine title and/or footing lines is allowed. The title and footing lines must be in sequence.

```
T1 'REPORT 1'
T2 'PAYROLL MASTER'
T3 'MONTHLY CENSUS' FOOTING
```

Title Contents

VISION:Results automatically centers the title contents in the report unless informed otherwise. Centering is based on the characters-per-line default of 132 unless overridden in the REPORT statement (such as REPORT 120 WIDE). The title contents (the text between single quotation marks including blanks, but excluding trailing blanks) are counted and subtracted from the characters per line and divided by 2. The result is the beginning print position for that title. For example:

```
REPORT 120 WIDE
T1 'TRIAL BALANCE REPORT'
```

The title TRIAL BALANCE REPORT would be placed at position 50 of the report line.

$$50 = (120 - 20) / 2$$

FIXED

If FIXED or FOOTING is used, it goes on the Tn statement of the TITLE/FOOTING line. Any and all TITLE/FOOTING identification statements have an implied (not stated) FIXED or FOOTING for the same line number.

To override the automatic centering of title or footing lines, supply the keyword FIXED in the title statement. The first character of the title contents appears in position 1 of the report line.

```
T1 'PAYROLL MASTER UPDATE REPORT' FIXED
```

The title contents begin at position 1 of the report line.

```
T1 '          TRIAL BALANCE REPORT' FIXED
```

Because there are 10 leading blanks in this title statement, the title begins at position 11 of the report line.

FOOTING

If FIXED or FOOTING is used, it goes on the Tn statement of the TITLE/FOOTING line. Any and all TITLE/FOOTING identification statements have an implied (not stated) FIXED or FOOTING for the same line number.

When the keyword FOOTING is added to a title statement, it identifies the line as a footing line, to be printed at the bottom of each page.

```
T1 'SUMMARY LINE COMPANY ABAZAB' FOOTING
```

The footing statement title contents are automatically centered and printed at the bottom of each page.

```
T1 'SUMMARY LINE COMPANY ABAZAB' FOOTING FIXED
```

Because the keyword FIXED has been added to the footing statement, the title begins at position 1 on the bottom of each page.

You can not specify FOOTING in an HTML report.

WITH n AFTER

The keyword phrase WITH n AFTER specifies the number of lines of spacing after the title or footing line.

Default spacing is WITH 1 AFTER. You can specify from 0 to 9 lines of spacing after the title or footing line.

```
T1 'TRIAL BALANCE' WITH 2 AFTER  
T1 'TRIAL BALANCE REPORT'  
T2 'QUARTERLY CONSENSUS' WITH 3 AFTER
```

You can use the WITH 0 AFTER if you want overstriking (such as highlighting) because spacing does not take place.

```
T1 'FREQUENCY DISTRIBUTION PAYROLL' WITH 0 AFTER
T2 'FREQUENCY DISTRIBUTION PAYROLL' WITH 2 AFTER
```

Title line 1 prints but spacing does not occur. Title line 2 then overstrikes title line 1 and double spaces.

Additional Information

If the VISION:Results program is producing a report and no title/footing statements or column headings are supplied, automatic page ejection, based on the number of lines per page, does not occur. This feature can be used if you are printing labels or continuous preprinted forms and do not want page ejection.

If you do not want any title/footing line but you do want page ejection (skip to channel 1 when the lines per page count reaches its maximum), you can supply a title/footing line with no title contents.

```
T1
```

VISION:Results writes a blank line for the above title statement and then a single space (remember the default for spacing is WITH 1 AFTER).

If you do not want the automatic spacing, you can specify the title statement in the following way:

```
T1 WITH 0 AFTER
```

In the following example, automatic centering of the title line with double spacing after is requested, as well as a fixed-print footing line with page ejection:

```
T1 'TITLE NUMBER 1' WITH 2 AFTER
T2 'FOOTING NUMBER 2' FIXED FOOTING
```

Title line 1 (T1) is automatically centered and printed with double spacing taking place after the line is printed. The footing line (T2) is printed, and ejection (skip to channel 1...top of next page) occurs.

To continue title or footing lines, enclose the value within quotation marks on a second or third line but do not repeat the command Tn, where n is 1 through 9. For example:

```
T1 'EMPLOYEE '
    'PAYROLL MASTER' WITH 2 AFTER
```

The result is the title:

```
EMPLOYEE PAYROLL MASTER
```

For example:

```
T1 'EMPLOYEE MASTER SUMMARY '
    'BY EACH DIVISION' FOOTING
```

The result is the footing line:

```
EMPLOYEE MASTER SUMMARY BY EACH DIVISION
```

In the next example, the title contains an apostrophe (single quotation mark). The title wanted is CONTROL LISTING FOR ABC AND CO'S ACCOUNTS RECEIVABLE. You can use the following two approaches:

- The single quotation mark approach:

```
T1 'CONTROL LISTING FOR ABC AND CO''S ACCOUNTS RECEIVABLE'
```

- The double quotation mark approach:

```
T1 "CONTROL LISTING FOR ABC AND CO'S ACCOUNTS RECEIVABLE"
```

If continuation to a second line is necessary, the following is acceptable.

- The single quotation mark approach:

```
T1 'CONTROL LISTING FOR ABC AND '  
   'CO''S ACCOUNTS RECEIVABLE'
```

- The double quotation mark approach:

```
T1 "CONTROL LISTING FOR ABC AND "  
   "CO'S ACCOUNTS RECEIVABLE"
```

- Or in combination:

```
T1 'CONTROL LISTING FOR ABC AND '  
   "CO'S ACCOUNTS RECEIVABLE"
```

```
T1 "CONTROL LISTING FOR ABC AND "  
   'CO''S ACCOUNTS RECEIVABLE'
```

Title or Footing Modification Statement

If you want to modify (insert variable information into) an existing title or footing line, you can do so by using the title/footing modification statement. This statement can never precede a T1 through T9 statement. The T1 through T9 must have been previously defined in your VISION:Results program.

The format of the title/footing modification statement is:

```
Tn+n {literal  
      {dataname  
      {reserved word}}
```

Figure 268 Format of the Title/Footing Modification Statement

Tn

T1 through T9 (Tn) indicates the title/footing line that is to have variable information inserted into it. The + (plus) and the number (n) indicate the beginning location of the new text (where it is to be inserted). Position 1 is relative to 1 of the title/footing line.

```
T1+10
```

The above title modification statement tells VISION:Results to place a data name, literal, or reserved word value starting at position 10 of title line 1.

Immediately following the location, enter the literal to be inserted, the data name contents to be inserted, or the reserved word contents to be inserted.

```
T1+10 'REPORT AR/110'  
T1+22 DESCRIPT  
T1+100 DYLPAGE
```

The first title modification is a literal, the second is a data name defined in an input file or work area, and the third is a reserved word containing the current page number.

The general rules for title modifications follow.

Alphanumeric Literal

Any data that is alphanumeric, including punctuation and special symbols, must be enclosed within single or double quotation marks. The maximum size of an alphanumeric literal is 30 characters.

```
T1+20 'ABC'  
T1+120 '123@ABC'  
T1+43 '*+;,XYZ@'
```

Any data that includes the single quotation mark (') or double quotation mark (") should be specified as follows:

T1+20 'ABC' 'DEF'	Two single quotation marks indicate a single quotation mark to be included. The result is ABC'DEF.
T1+20 'ABC"DEF'	Double quotation mark to be included. The result is ABC"DEF.
T1+20 "ABC'DEF"	Single quotation mark to be included. The result is ABC'DEF.
T1+20 "ABC" "DEF"	Single and double quotation marks to be included. The result is ABC"DEF.

Numeric Literal

Any data that is numeric only can be specified as follows:

T1+20 1234567	Value of 1234567 is to be inserted.
T1+10 00010	Value of 00010 is to be inserted.

Data Names

Data names defined in the FILE field definitions or work areas can also be used when modifying a title or footing line. The contents of the data name (that is, its value) are inserted at the location specified.

T1+20 EMPNAME The contents of EMPNAME are inserted.

T1+88 COMMENT The contents of COMMENT are inserted.

Each data name's data is edited according to the field definition provided. VISION:Results packs and unpacks the field if necessary.

Reserved Words

Reserved words can also be used to modify a title or footing line. The contents of the reserved word (that is, its value) are inserted at the location specified.

T1+1 DYLDATE The MM/DD/YY contents of DYLDATE are inserted.

T1+120 DYLPAGE The contents of DYLPAGE (the current page count value) are inserted.

For a complete list of reserved words, including those containing different formats of date and page, see [Appendix A: Reserved Words](#).

Additional Information

The title modification statement is never executed during VISION:Results procedure logic. To dynamically change part of a title during execution, the data name into which contents are to be inserted into the title must be modified in the procedure logic of the VISION:Results program.

[Figure 269](#) demonstrates overrides in the REPORT and title modification statements.

```
REPORT 100 WIDE 55 LONG
T1 'EMPLOYEE STATEMENT' WITH 4 AFTER
T1+71 DYLPAGE
```

Figure 269 Overrides in the REPORT and Title Modification Statements

With a 100 characters-per-line report, the centered title begins in absolute position 42 of the report. The additional text, DYLPAGE, prints with its beginning character in position 71 of the title line.

The following example shows title modification and footing line generation.

```
REPORT
T1 'EMPLOYEE STATEMENT' WITH 4 AFTER
T1+80 DYLPAGE
T2 'PAYROLL MASTER REPORT' FOOTING
```

Figure 270 Title Modification and Footing Line Generation

With all defaults taken on the REPORT command (which is not required in our example), the centered title begins at absolute position 57 (remember that the default characters-per-line is 132). The additional text, DYLPAGE, prints with its beginning character in position 80 of the title line. The PAYROLL MASTER REPORT footing line begins at position 55 and prints as the last line at the bottom of each page of the report.

```
T1 'EMPLOYEE STATEMENT' WITH 4 AFTER
T1+70 DYLPAGE
T2 'PAYROLL MASTER REPORT' FOOTING
T2+1 DYLDATE
```

```
or
T1 'EMPLOYEE STATEMENT' WITH 4 AFTER
T2 'PAYROLL MASTER REPORT' FOOTING
T1+70 DYLPAGE
T2+1 DYLDATE
```

Figure 271 REPORT Command Example

In the following example, spacing is required before printing the title line.

```
REPORT 120 WIDE
T1 WITH 3 AFTER
T2 'PAYROLL MASTER UPDATE REPORT' WITH 2 AFTER
```

Figure 272 Example of Spacing

The contents of title statement 2 (T2) are printed on the third line after title 1.

The example below shows how to insert variable information into a report title (the first title line) that requires only today's date.

```
T1 WITH 0 AFTER
T1+120 DYLDATE
```

Figure 273 Variable Information

Limitations and Constraints

- Reserved words and your data names cannot be used in title or footing statements.
- The phrase WITH n AFTER is not valid in title or footing modification statements. To indicate WITH n AFTER, code as follows:

```
T1 'REPORT 01' WITH 2 AFTER
T1+1 DYLDATE
T2 'REPORT 02' WITH 3 AFTER
T2+120 'REPORT 320-1'
```

Figure 274 WITH n AFTER Code

- Title and footing modification statements cannot be subordinate to an IF statement.

- The print location specified in a title or footing modification statement and the length of the literal or the size of the data name or reserved word cannot be greater than the number of characters per line.

```
REPORT 145 WIDE  
T1 'REPORT TITLE 1'  
T1+132 DYLPAGE  
  
REPORT 150 WIDE  
T1 'REPORT TITLE 2'  
T1+120 DYLDATAPAG  
  
REPORT 130 WIDE  
T1 'REPORT TITLE 3' WITH 2 AFTER  
T1+120 DYLDAT
```

Figure 275 Print Location Examples

- The last position of the print line, whether a detail line or a title line, is not available. It is reserved in case a minus sign must be placed there.

Whenever you change the contents of the modification field, the change does not take effect until page headings or footings are next printed. Complete report examples are shown at the end of this chapter. At least one example demonstrates how to modify the contents of a data name that is being used in a title modification statement.

LIST Statement

The LIST statement specifies that a report line is to be printed and indicates what the contents of the line are to be.

VISION:Results automatically composes the report line (including determining column heading locations and print field locations) so that the fields are properly positioned on the page and print in the order specified. The width of each column is determined by whichever is larger:

- The column heading of the field being printed
- The field size as defined
- The edit mask used by VISION:Results, if the field is NU, BI, or PD

You can optionally format your report layout using the fixed print position facility of LIST.

Following is the syntax for all the keywords and edit codes that can be used within a LIST statement. See the individual sections in this chapter for details on each keyword. For complete edit code descriptions, see [Edit Codes on page 313](#).

```
LIST ![{SUPPRESS | PRIOR | SUM | SUMn}] [(| !{dataname | (index) | literal | TALLY }
  [{E | A | Z | B | P | D | S}] [NB | Q]
  [{W | NP} | {U | NZ} | {Y | NE} | {X | NA}
  [{F | DE} | {G | NDE} | {K | DNE} | {H | DA} | {J | NDA} | {L | DNA} | {D4 | V}]
  [AT {n | dataname [+n]} ]
  [(column heading)] !...[] !...
  [WITH [{EJECT | n} BEFORE] [{AND} [{EJECT | n}] AFTER]]
```

Figure 276 Keywords and Edit Codes That Can Be Used Within a LIST

The default is 0 before and 1 after.

```
LIST NAME
LIST SUM BALANCE WITH 2 AFTER
LIST 'NUMBER OF RECORDS PRINTED' AT ACCOUNT TALLY AT
  BALANCE
LIST SUPPRESS ACCOUNT TRANS NAME BALANCE E JBALANCE
  E COMMENT
LIST BALANCE (BALANCE'AMOUNT)
LIST NAME AT 5 ADDRESS AT 35 WITH 2 AFTER
LIST PRIOR (ACCT NAME)
LIST FIELD1 (INX) (HEADER1) FIELD2
```

LIST Command

The LIST command begins each LIST statement. The remaining items of the statement follow the command and are positional—they must appear in the order listed:

```
LIST dataname1 [INX] [datanamen] ...
  [WITH {n} BEFORE AND {n} AFTER]
      {EJECT}
```

Figure 277 LIST Command

Each data name in the LIST statement gives information concerning a single field to be printed on the report. The order, from left to right, of data names in the LIST statement is the order in which these fields appear on the printed report. For example:

```
LIST NAME EMPNO NET
```

This tells VISION:Results to print a report line listing name, employee number, and net pay for each record read.

At least one data name or literal must appear in the LIST statement.

Indexed data names can be entered in the LIST statement. For example:

```
FILE FILE1 FB 80
RECORD1 80
FIELD1 5 1
FIELD2 5
FIELD3 5

INX - 0

LIST FIELD1 (INX) (HEADER1) FIELD2

FIN
```

Figure 278 Indexed Data Names

The LIST command can be executed at detail, ON CHANGE IN, or ON FINAL time. You can GOTO or PERFORM (to execute) a LIST. However, you cannot GOTO or PERFORM a detail LIST from ON CHANGE IN or ON FINAL logic.

LIST statements not governed by (subordinate to) an ON CHANGE IN or ON FINAL statement are said to be detail lines. LIST statements subordinate to ON END OF INPUT and ON END OF SORTING are also said to be detail lines. For example:

```
LIST DIV NAME EMPNO NET      ← detail line
ON CHANGE IN DIV
  LIST SUM NET WITH 2 BEFORE 2 AFTER
ON FINAL
  LIST SUM NET WITH 2 BEFORE
```

Figure 279 ON CHANGE IN or ON FINAL Statement

The LIST command can never be subordinate to an IF statement at control break time (ON CHANGE IN or ON FINAL). For example:

```
ON CHANGE IN ACCOUNT
IF ACCOUNT EQ 'WO'
  MOVE '-' TO COMMENT
ENDIF
LIST COMMENT SUM BALANCE
```

Figure 280 IF Statement

The LIST command can never be selectively ignored at control break time (ON CHANGE IN or ON FINAL).

A validation error is generated if a control break field is indexed in a LIST statement. To avoid this situation, move your control break field to a work area and then list it.

Data Name Format

Each data name in the LIST statement identifies a field to be printed. It can also specify any editing to be done on the field and/or operations to be performed on the field, that is, totaling of the field contents and placing the printed output in a designated location.

Each data name is positional and is of the following form:

Format 1:

```
LIST [keyword] dataname [edit code] [column heading]
LIST [keyword] dataname [edit code] [column heading] AT dataname
LIST [keyword] dataname [edit code] [column heading] AT dataname + n
LIST [keyword] dataname [edit code] AT nnn
```

Figure 281 Data Name Format 1

Format 2:

```
LIST literal
LIST literal AT dataname
LIST literal AT dataname + n
LIST literal AT nnn
```

Figure 282 Data Name Format 2

Format 3:

```
LIST TALLY [edit code] [column heading]
LIST TALLY [edit code] [column heading] AT dataname
LIST TALLY [edit code] [column heading] AT dataname + n
LIST TALLY AT nnn
```

Figure 283 Data Name Format 3

See [Edit Codes on page 313](#) for edit code descriptions.

Keyword

The keyword specifies an operation to be performed on the data name field. Only one of the following keywords can be selected:

- [SUM](#)
- [SUPPRESS](#)
- [PRIOR](#)
- [SUM1 Through SUM6](#)

SUM

The keyword SUM is valid only during control break (ON CHANGE and ON FINAL) processing. It tells VISION:Results to total the contents of the data name field specified up to the current control break.

If there is detail printing, VISION:Results adds all fields that have to be totaled to appropriate internal accumulators whenever the first physical detail VISION:Results LIST statement is executed.

If there are no detail LIST statements, VISION:Results totals all required fields whenever it exits the automatic cycle or an ACCEPT command is executed.

Following printing, the total is added to the next highest level of control break (if summed) and then cleared to zero. Then, the next set of values begins accumulating until the next change in the control break field, when printing and clearing occur again. The data name (field) qualified by the keyword SUM must contain numeric, packed, or binary data.

Examples of valid formats are:

```
ON CHANGE IN DEPARTMENT
  LIST SUM NETPAY

ON CHANGE IN DEPARTMENT
  LIST SUM NETPAY SUM GROSSPAY

ON FINAL
  LIST SUM NETPAY SUM GROSSPAY SUM HOURS
```

If the keyword SUM refers to more than one field, you can code the LIST in the following form:

```
ON CHANGE IN DEPARTMENT
  LIST SUM (NETPAY GROSSPAY)

ON FINAL
  LIST SUM (NETPAY GROSSPAY HOURS)
```

The parentheses must enclose those data names that are to be operated on by the keyword SUM. Note that the individual sums of NETPAY, GROSSPAY, and HOURS are to be printed individually and not accumulated together.

The keyword SUM and its data name can be used in an arithmetic expression, if required, during control break logic (ON CHANGE IN or ON FINAL).

```
TOTAL=SUM NETPAY + TOTAL
ACCUM=SUM NETPAY+SUM OVERTIME
TOT=1.10 * SUM NETPAY
```

The keyword SUM can never be used during detail processing or detail listing. When SUM is used, the data name being summed cannot be indexed.

The following example shows how to produce a report containing the employee names listed by the department number, the net pay for each employee, and the total of net pay received by employees in each department.

To print the total net pay for each department, specify the department number as the trigger for the first control break (see [ON CHANGE IN on page 409](#)), then qualify the net pay (NETPAY) field with the keyword SUM.

The statement causes the net pay of all employees in the first department to be totaled, then the net pay of employees in the second department is totaled, and so on. It is assumed that the file is in department number sequence and that the column headings and edit formats were defined in the FILE field definitions.

```
CONTROL DEPARTMENT
LIST SUPPRESS DEPARTMENT EMPNAME NETPAY
ON CHANGE IN DEPARTMENT
  LIST SUM NETPAY WITH 2 BEFORE AND 2 AFTER
```

Figure 284 Keyword SUM Example 1

DEPARTMENT	EMPLOYEE NAME	NET PAY
1	JOHN JONES	865.98
	MARY POLLACK	987.88
	STEVEN SMITH	678.98
		2532.84
2	ALBERT AMES	587.23
	JOHN CARLS	989.90
		1577.13
3	ANITA BAIRD	667.77
	FRANK KOCH	348.55
		1016.32

Figure 285 Keyword SUM Output 1

Using the same example, to print the total net pay for all departments, again specify department (DEPARTMENT) as the control break, then qualify the net pay (NETPAY) with the keyword SUM at both ON CHANGE IN and ON FINAL time. Assume that the column headings and edit formats were defined in the FILE field definitions.

```
CONTROL DEPARTMENT
LIST SUPPRESS DEPARTMENT EMPNAME NETPAY
ON CHANGE IN DEPARTMENT
    LIST SUM NETPAY WITH 2 BEFORE AND 2 AFTER
ON FINAL
    LIST SUM NETPAY WITH 2 BEFORE
```

Figure 286 Keyword SUM Example 2

DEPARTMENT	EMPLOYEE NAME	NET PAY
1	JOHN JONES	865.98
	MARY POLLACK	987.88
	STEVEN SMITH	678.98
		2532.84
2	ALBERT AMES	587.23
	JOHN CARLS	989.90
		1577.13
3	ANITA BAIRD	667.77
	FRANK KOCH	348.55
		1016.32
		5126.29

Figure 287 Keyword SUM Output 2

SUPPRESS

Using SUPPRESS tells VISION:Results to print the contents of the field following the keyword SUPPRESS only if the field's value has changed. However, when a page eject or a higher-level control break occurs, the field is printed.

Valid formats for the keyword SUPPRESS are:

```
LIST SUPPRESS DEPARTMENT
LIST SUPPRESS DIVISION SUPPRESS DEPARTMENT
```

If the keyword SUPPRESS (like the other qualifiers SUM and PRIOR) refers to more than one field, you can code the LIST in the following form:

```
LIST SUPPRESS (DIVISION DEPARTMENT)
```

Valid formats for the keyword SUPPRESS with column headings as part of the LIST statement are:

```
LIST SUPPRESS (DIVISION (DIV'NO) DEPARTMENT)
LIST SUPPRESS (DIVISION (DIV'NO) DEPARTMENT (DEPT))
LIST SUPPRESS (DIVISION DEPARTMENT (DEPT))
```

See [Column Heading on page 398](#).

The keyword SUPPRESS is not valid at ON FINAL time.

You cannot use LIST statements with SUPPRESS and DELIM; VISION:Results produces an error message.

For example, if you do not qualify ACCOUNT with the keyword SUPPRESS as shown in the following LIST statement:

```
LIST ACCOUNT TRANS NAME BALANCE
```

the report looks like:

ACCOUNT	TRANS	NAME	BALANCE
BO	8006547	TOWNSON	94.99
EO	6002587	NELLISEN	15.00
EO	7082509	REED	5.00
EO	6208657	SMYTH	32.00
FO	6107265	KRUSE	43.00
FO	8011508	ALHVERS	178.76
FO	6024963	CHARLES	3.80
FO	6044395	CORWIN	45.24
GO	6410905	LOUIS	.00

Figure 288 Qualify ACCOUNT

With the data name ACCOUNT qualified with the keyword SUPPRESS:

```
LIST SUPPRESS ACCOUNT TRANS NAME BALANCE
```

the resulting report looks like:

ACCOUNT	TRANS	NAME	BALANCE
BO	8006547	TOWNSON	94.99
EO	6002587	NELLISEN	15.00
	7082509	REED	5.00
	6208657	SMYTH	32.00
FO	6107265	KRUSE	43.00
	8011508	ALHVERS	178.76
	6024963	CHARLES	3.80
	6044395	CORWIN	45.24
GO	6410905	LOUIS	.00

Figure 289 Qualify ACCOUNT Data Name

PRIOR

Qualifying the field with the keyword PRIOR instructs VISION:Results to print the contents of the field from the previous record, not the one currently in the input record area.

This refer back option is very useful when descriptions, department numbers, and so on, are to be printed on control break lines and you want to refer back to the prior record rather than the one that caused the break that is currently in the record area.

The keyword PRIOR cannot be used in a detail-time LIST statement, nor can the keyword be used in a MOVE or IF statement. In fact, the keyword PRIOR can only be used in a LIST statement during control break processing:

```
LIST SUPPRESS DIV DEPT NAME NETPAY
ON CHANGE IN DIV
  LIST PRIOR DEPT SUM NETPAY
```

Figure 290 Keyword PRIOR Use

Valid formats of the MOVE PRIOR and IF PRIOR do not exist. If the prior account value as in the MOVE PRIOR and IF PRIOR is required, the following can be used:

```
ON ONE
  MOVE ACCOUNT TO SAVEACCT
ENDONE
.
.
detail processing logic
.
.
ON CHANGE IN DIV
  IF ACCOUNT EQ X'FF' GOTO TAGA ENDIF
  MOVE SAVEACCT TO WACCT
  MOVE ACCOUNT TO SAVEACCT
TAGA:
LIST .....
```

Figure 291 Working with MOVE PRIOR and IF PRIOR

In this example, one-time only logic is used (ON ONE/ENDONE) to save the first record's ACCOUNT value in a work area. At control break time, the work area SAVEACCT can be examined (it is the prior account code). To re-establish the new prior account code (for the next control break), the ACCOUNT field value must be moved to SAVEACCT at this time.

If the field is being totaled (SUM), the keyword PRIOR cannot be used because VISION:Results obtains the value of a totaled field from an internal accumulator.

If the field is specified as a control break, you need not use the keyword PRIOR. At control break time, VISION:Results automatically refers to the preceding record's field if that data name is referenced as a control break in the CONTROL statement.

Valid formats for PRIOR are:

```
CONTROL DIVISION
LIST SUPPRESS(DIVISION DEPT LOC) SALESMAN SALES
ON CHANGE IN DIVISION
  LIST PRIOR DEPT DIVISION
```

```
CONTROL DIVISION
LIST SUPPRESS(DIVISION DEPT LOC) SALESMAN SALES
ON CHANGE IN DIVISION
  LIST PRIOR DEPT PRIOR LOC
```

```
CONTROL DIVISION
LIST SUPPRESS(DIVISION DEPT LOC) SALESMAN SALES
ON CHANGE IN DIVISION
  LIST DIVISION PRIOR DEPT
```

Figure 292 Formats for PRIOR

If the keyword PRIOR refers to more than one field, you can code the LIST in the following form:

```
CONTROL DIVISION
ON CHANGE IN DIVISION
  LIST PRIOR (DEPT LOCATION)
```

```
CONTROL DIVISION
ON CHANGE IN DIVISION
  LIST DIVISION PRIOR (DEPT LOCATION)
```

Figure 293 Keyword PRIOR in More Than One Field

SUM1 Through SUM6

The keyword SUMn tells VISION:Results to accumulate the total amount of the field specified in the data name through the control break level that is specified by the integer. That is, the total is not added to the next level nor is it cleared to zero until the next higher control break occurs. For example:

```
LIST SUM1 NETPAY
```

This statement tells VISION:Results to total the net pay field and not clear it until the second control break level has been detected.

SUMn cannot exceed the available control breaks as specified in the CONTROL statement.

If you print a report like the one described for the keyword SUM, but use SUM1 instead, the result is the following:

```
CONTROL DEPARTMENT
LIST SUPPRESS DEPARTMENT EMPNAME NETPAY
ON CHANGE IN DEPARTMENT
  LIST SUM1 NETPAY WITH 2 BEFORE AND 2 AFTER
```

Figure 294 Using SUM1 Example

DEPARTMENT	EMPLOYEE NAME	NET PAY
1	JOHN JONES	865.98
	MARY POLLACK	987.88
	STEVEN SMITH	678.98
		2532.84
2	ALBERT AMES	587.23
	JOHN CARLS	989.90
		4109.97
3	ANITA BAIRD	667.77
	FRANK KOCH	348.55
		5126.29

Figure 295 Using SUM1 Results

Assume that all column headings and edit formats are defined in the field definitions of the FILE statement.

Data Name

The data name provided in the LIST statement is the field that is to be printed.

The data name can be entered in the LIST in three forms:

```

dataname [edit code]
dataname [edit code] AT dataname2
dataname [edit code] AT dataname2 + n

```

Figure 296 Data Name in Three Forms

See [Edit Codes on page 313](#) for edit code definitions.

There is a fourth form of data name (dataname AT n) used with fixed print positioning. For information on this fourth form, see [Fixed Print Position Reporting on page 406](#).

You can override or specify data name edit codes in your LIST statement. Edit codes are defined in the *Advantage VISION:Results for z/OS Getting Started*. To edit a field, place the edit code immediately after the name of the field you want to edit. For example:

```
LIST NAME ADDRESS SOCSEC S
```

The edit code in the LIST statement causes the Social Security Number (SOCSEC) to be printed with two hyphens, for example, 999-99-9999.

The edit codes specified in a LIST statement apply only to that statement. No subsequent LIST statements are affected.

If your program contained the following two LIST statements, the date field would be edited only when the first LIST statement is executed.

```
.  
. LIST NAME ACCT BALANCE A DATE D  
. ON CHANGE IN DEPT  
  LIST DEPT DATE  
. .
```

Figure 297 Two LIST Statements

To assure that editing takes place at all levels, respecify the edit code after the data name on each LIST statement. Using the same example, specify the edit code a second time on the second LIST statement.

```
.  
. LIST NAME ACCT BALANCE A DATE D  
. ON CHANGE IN DEPT  
  LIST DEPT DATE D  
. .
```

Figure 298 Two LIST Statements Simplified

If you have specified an edit code in a field definition, you can override it by using a different edit code for the same field in a LIST statement. For example:

```
FILE ARFILE FB 352 5280  
  ACCT 2 182  
  NAME 25 85  
  TRANS 7 4  
  BALANCE 5 170 PD 2 A  
  DATE 6 44 NU D  
. LIST ACCT BALANCE DATE  
. ON CHANGE IN ACCT  
  LIST NAME BALANCE E
```

Figure 299 Two LIST Statements with Edit Code

In this example, if BALANCE is zero, it prints as .00 upon execution of the first LIST statement (an A edit code applies) and as blanks upon execution of the second LIST statement (an E edit code applies).

Specifying an edit code in a LIST statement is useful if you need to edit a field that cannot be edited in the field definition, as is the case when you are using COBOL data definitions in your program (see [Chapter 15: Using the COPY Commands](#)) or when you want to override an edit code specified in a field definition statement.

The LIST data name tells VISION:Results to automatically position the fields on the report line from left to right.

```
LIST DIVISION BRANCH DEPARTMENT ← detail line 1
LIST ACCOUNT TRANS           ← detail line 1
LIST NAME ADDRESS CITY STATE ZIP ← detail line 1
```

Figure 300 LIST Data Name

Assuming that each of the LIST statements above is in three different programs, the fields supplied in each LIST print left to right in the order specified. Column headings (as defined or their defaults) would be supplied for all data names listed.

When a data name is used in more than one LIST statement, additional occurrences of the data name (field) are aligned at the same location on the report line unless you specify otherwise (for example, AT or AT +n). For example:

```
CONTROL ACCOUNT
LIST ACCOUNT TRANS NAME BALANCE
ON CHANGE IN ACCOUNT
LIST SUM BALANCE WITH 2 BEFORE AND 2 AFTER ←
```

ACCOUNT	TRANS	NAME	BALANCE	
BO	8006547	TOWNSON	94.99	← detail line 1
			94.99	← control line 1
EO	6002587	NELLISEN	15.00	← detail line 1
EO	7082509	REED	5.00	
EO	6208657	SMYTH	32.00	
			52.00	← control line 1

Figure 301 Data Name Using More Than One LIST Statement

Detail line 1 was produced using the LIST ACCOUNT TRANS NAME BALANCE statement while control line 1 was produced using LIST SUM BALANCE. Note that BALANCE is aligned under the column heading BALANCE in both cases.

Another example:

```
CONTROL ACCOUNT
LIST SUPPRESS ACCOUNT TRANS NAME BALANCE
ON CHANGE IN ACCOUNT
LIST ACCOUNT SUM BALANCE WITH 2 BEFORE AND 2 AFTER ""
```

ACCOUNT	TRANS	NAME	BALANCE	
BO	8006547	TOWNSON	94.99	← detail line 1
BO			94.99	← control line 1
EO	6002587	NELLISEN	15.00	← detail line 1
	7082509	REED	5.00	
	6208657	SMYTH	32.00	
EO			52.00	← control line 1

Figure 302 Data Name Using More Than One LIST Statement Example 2

Both ACCOUNT and BALANCE are aligned under their respective column headings at control line 1. Note the keyword, SUPPRESS, added to ACCOUNT. Also note that, because ACCOUNT is specified on the CONTROL statement, the use of PRIOR is not required on the ON CHANGE IN ACCOUNT LIST statement.

This example demonstrates how VISION:Results can automatically position a data name that is not specified in a detail line 1.

```

CONTROL ACCOUNT
LIST SUPPRESS ACCOUNT TRANS NAME BALANCE
ON CHANGE IN ACCOUNT
  LIST ACCOUNT SUM BALANCE PRIOR SALESMAN
    WITH 2 BEFORE AND 2 AFTER

```

ACCOUNT	TRANS	NAME	BALANCE	SALESMAN	
BO	8006547	TOWNSON	94.99		← detail line 1
BO			94.99	OLDTIMER	← control line 1
EO	6002587	NELLISEN	15.00		
	7082509	REED	5.00		
	6208657	SMYTH	32.00		
EO			52.00	MCATEE	← control line 1

Figure 303 Data Name Using More Than One LIST Statement Example 3

Because the ON CHANGE IN ACCOUNT (control break line 1) has a data name never before specified on a previous LIST statement, VISION:Results automatically positions the field to the right of the last field positioned (BALANCE), with its column heading.

In this example, ON FINAL logic is added:

```

CONTROL ACCOUNT
LIST SUPPRESS ACCOUNT TRANS NAME BALANCE
ON CHANGE IN ACCOUNT
  LIST ACCOUNT SUM BALANCE PRIOR SALESMAN
    WITH 2 BEFORE AND 2 AFTER
ON FINAL
  LIST SUM BALANCE WITH 2 BEFORE

```

ACCOUNT	TRANS	NAME	BALANCE	SALESMAN	
BO	8006547	TOWNSON	94.99		← detail line 1
BO			94.99	OLDTIMER	← control line 1
					control line 1
EO	6002587	NELLISEN	15.00		
	7082509	REED	5.00		
	6208657	SMYTH	32.00		
EO			52.00	MCATEE	← control line 1
					control line 1
			146.99		← final line 1

Figure 304 Data Name Using More Than One LIST Statement Example 4

The final total of BALANCE is aligned properly.

dataname AT dataname2

The LIST statement `dataname AT dataname2` tells VISION:Results to use the automatically positioned field location of `dataname2` and print `dataname` at that location also. This form is generally used at control break time (ON CHANGE IN or ON FINAL) and can be used for multi-line printing (see [Alignment of Data Names and Literals at Detail Time \(Multi-line\) on page 402](#) and [Alignment of Data Names and Literals at ON CHANGE IN and ON FINAL Time on page 403](#)). For example:

```
CONTROL ACCOUNT
LIST SUPPRESS ACCOUNT NAME TRANS BALANCE
ON CHANGE IN ACCOUNT
  LIST PRIOR SALESMAN AT NAME SUM BALANCE          ←
    WITH 2 BEFORE AND 2 AFTER
ON FINAL
  LIST SUM BALANCE WITH 2 BEFORE
```

ACCOUNT	NAME	TRANS	BALANCE	
BO	TOWNSON	8006547	94.99	← detail line 1
	J OLDTIMER		94.99	← control line 1
EO	NELLISEN	6002587	15.00	
	REED	7082509	5.00	
	SMYTH	6208657	32.00	
	PHILPOTT	2093831	100.00	
	HEATHER ANN MCATEE		152.00	← control line 1
			246.99	← final line 1

Figure 305 Dataname AT Dataname2

When using `dataname AT`, you are in control. VISION:Results does not decimal align the field, nor does it examine the field to see if its length is greater than the field to which it is positioning.

The example below produces erroneous report output:

```
CONTROL ACCOUNT
LIST SUPPRESS ACCOUNT NAME TRANS BALANCE
ON CHANGE IN ACCOUNT
  LIST PRIOR SALESMAN AT TRANS SUM BALANCE          ←
    WITH 2 BEFORE AND 2 AFTER
ON FINAL
  LIST SUM BALANCE WITH 2 BEFORE
```

Figure 306 Dataname AT Dataname2 Example 2

There is an 11-byte difference in the length of TRANS (7) and SALESMAN (18). VISION:Results builds the print line from right to left. First, it moves in SUM BALANCE (control line 1), followed by SALESMAN AT TRANS. In the first control line print, the BALANCE amount printed was very small. However, in the second control line print, the high order digits of the BALANCE value are overlaid.

ACCOUNT	NAME	TRANS	BALANCE	
BO	TOWNSON	8006547	94.99	← detail line 1
		J OLDTIMER	94.99	← control line 1
EO	NELLISEN	6002587	15.00	
	REED	7082509	5.00	
	SMYTH	6208657	32.00	
	PHILPOTT	2093831	100.00	
		HEATHER ANN MCATEE	52.00	← control line 1
			246.99	← final line 1

Figure 307 Dataname AT Dataname2 Example 2 Results

dataname AT dataname2 +n

Dataname AT dataname2 +n is similar to the previous definition and should be used only when you are refining your program's output report.

LIST dataname AT dataname2 +n tells VISION:Results to use the automatically positioned field location of dataname2, add the displacement (maximum 99) to it, and print dataname. VISION:Results does not provide column headings for data names that are given specific alignment, unless the column heading is specified in the LIST statement. This form can be used at detail time, during multi-line printing, or at control break time. For example:

```
LIST SUPPRESS ACCOUNT TRANS BALANCE
ON CHANGE IN ACCOUNT
  LIST SUM BALANCE AT BALANCE + 10          ←
  WITH 2 BEFORE AND 2 AFTER
```

ACCOUNT	TRANS	BALANCE	
BO	8006547	94.99	← detail line 1
		94.99	← control line 1
EO	6002587	15.00	← detail line 1
	7082509	5.00	
	6208657	32.00	
		52.00	← control line 1

Figure 308 Dataname AT Dataname2 +n

There is no column heading for BALANCE AT BALANCE + 10 because you specified the alignment.

The example below uses a column heading with a dataname AT +n:

```
LIST SUPPRESS ACCOUNT TRANS BALANCE
ON CHANGE IN ACCOUNT
  LIST SUM BALANCE AT BALANCE +10 (TOTAL)      ←
  WITH 2 BEFORE AND 2 AFTER
```

ACCOUNT	TRANS	BALANCE	TOTAL	
BO	8006547	94.99		
			94.99	← control line 1
EO	6002587	15.00		
	7082509	5.00		
	6208657	32.00		
			52.00	← control line 1

Figure 309 Dataname AT Dataname2 +n Example 2

For example:

```
LIST CONSTANT CONSTANT AT CONSTANT + 50
```

This is the only way you can repeat a data name on the same LIST statement. Note that the second entry of CONSTANT is AT CONSTANT plus 50.

If you had typed it as:

```
LIST CONSTANT CONSTANT
```

VISION:Results positions the second CONSTANT in the same position as the first CONSTANT. Remember that when a data name is used once, a second entry is automatically aligned to the position calculated for the first entry.

For example:

```
LIST EMPLOYEE ADDRESS AT EMPLOYEE +45 (ADDRESS)
```

The data name EMPLOYEE is automatically positioned by VISION:Results (centered) with a column heading. ADDRESS is displaced 45 positions to the right of EMPLOYEE with the appropriate column heading. ADDRESS was displaced and not automatically positioned (centered) on the report line.

The resulting report might appear awkward, and the technique could produce overlapping of field values.

Literal

You can type literal data into a LIST statement, enclosed in single quotation marks (''). Literals are normally used during control break printing; however, they can also be used at detail printing. The maximum length of a literal in a LIST statement is 30 characters.

The literal can be entered in the LIST statement in the following three formats:

```
literal
literal AT dataname
literal AT dataname + n
```

Figure 310 Literal Three Formats

There is a fourth format used when fixed print position reporting is wanted. For information on this fourth format, see [Fixed Print Position Reporting on page 406](#).

The definitions of dataname and dataname + n are equivalent to those defined in the previous section on data names and are not described in detail here.

If you enter a literal in a LIST statement, you cannot give it a column heading. However, you can list a literal that also has a column heading by defining the literal with a column heading in a work area and using its data name in the LIST statement. For example:

```
WORKAREA CHECKOFF 3 (CHECK'OFF) VALUE '-'
.
.
LIST ACCOUNT NAME BALANCE CHECKOFF
```

Figure 311 Literal in a LIST Statement

Following are three examples showing a control break and printing of the literal ACCOUNT TOTAL using the three forms available.

```
CONTROL ACCOUNT
LIST SUPPRESS ACCOUNT TRANS NAME BALANCE
ON CHANGE IN ACCOUNT
  LIST 'ACCOUNT TOTAL' SUM BALANCE          ←
    WITH 2 BEFORE AND 2 AFTER
```

ACCOUNT	TRANS	NAME	BALANCE
BO	8006547	TOWNSON	94.99
control line 1 →		94.99	ACCOUNT TOTAL
EO	6002587	NELLISEN	15.00
	7082509	REED	5.00
	6208657	SMYTH	32.00
control line 1 →		52.00	ACCOUNT TOTAL

Figure 312 Control Break and Printing of Literal ACCOUNT TOTAL Example 1

Because detail and control lines are entities unto themselves, the literal ACCOUNT TOTAL forces VISION:Results to create its own column. Note that there is no column heading.

The second example is:

```
CONTROL ACCOUNT
LIST SUPPRESS ACCOUNT TRANS NAME BALANCE
ON CHANGE IN ACCOUNT
  LIST 'ACCOUNT TOTAL' AT ACCOUNT SUM BALANCE      ←
    WITH 2 BEFORE AND 2 AFTER
```

Figure 313 Control Break and Printing of Literal ACCOUNT TOTAL Example 2

The resulting output looks like this:

ACCOUNT	TRANS	NAME	BALANCE	
BO	8006547	TOWNSON	94.99	
ACCOUNT TOTAL			94.99	← control line 1
EO	6002587	NELLISEN	15.00	
	7082509	REED	5.00	
	6208657	SMYTH	32.00	
ACCOUNT TOTAL			52.00	← control line 1

Figure 314 Control Break and Printing of Literal ACCOUNT TOTAL Results 2

Because the literal ACCOUNT TOTAL is greater in length than either the column heading ACCOUNT or the field length of ACCOUNT (which is 2), VISION:Results automatically left-aligned the literal with the column heading.

The third example is:

```
CONTROL ACCOUNT
LIST SUPPRESS ACCOUNT TRANS NAME BALANCE
ON CHANGE IN ACCOUNT
  LIST 'ACCOUNT TOTAL' AT ACCOUNT+5 SUM BALANCE    ←
    WITH 2 BEFORE AND 2 AFTER
```

Figure 315 Control Break and Printing of Literal ACCOUNT TOTAL Example 3

The resulting output looks like this:

ACCOUNT	TRANS	NAME	BALANCE	
BO	8006547	TOWNSON	94.99	
ACCOUNT TOTAL			94.99	← control line 1
EO	6002587	NELLISEN	15.00	
	7082509	REED	5.00	
	6208657	SMYTH	32.00	
ACCOUNT TOTAL			52.00	← control line 1

Figure 316 Control Break and Printing of Literal ACCOUNT TOTAL Results 3

ACCOUNT+5 identifies the field under which the literal is to be printed.

TALLY / TALLYn

The keywords TALLY and TALLYn can be used in an arithmetic statement, or they can be printed out. TALLY is the total number of records reported for this control break level. That is, if there is detail printing, TALLY is the count of the number of times the first detail LIST statement in the program is executed.

Use TALLYn if you want, for emphasis or ease of identification, each TALLY amount for multiple control breaks printed in a unique column. This syntax automatically aligns the TALLY amount in a unique column for each break level. You can have up to six break levels (TALLY1 through TALLY6).

If there is no detail printing, TALLY is the count of the number of times the procedure logic exited the program using the automatic cycle, or an ACCEPT command was executed for a given control break. TALLY is valid only during control break processing. TALLY can never be qualified with the SUM, PRIOR, SUPPRESS, or SUMn keywords.

TALLY can be entered in the LIST statement in three forms:

```
TALLY
TALLY AT dataname2
TALLY AT dataname2 + n
```

Figure 317 TALLY Three Forms

There is a fourth form, which can be found in [Fixed Print Position Reporting on page 406](#).

The definitions of dataname2 and dataname2 + n are equivalent to those defined in the previous section on data names and are not described here.

For example, you can use TALLY to print the number of transactions per account and the overall number of accounts processed.

```
LIST ACCOUNT TRANS NAME BALANCE
ON CHANGE IN ACCOUNT
  LIST SUM BALANCE TALLY WITH 2 BEFORE AND 2 AFTER
ON FINAL
  LIST SUM BALANCE TALLY WITH 2 BEFORE
```

ACCOUNT	TRANS	NAME	BALANCE	TALLY
BO	8006547	TOWNSON	94.99	
control line 1 →			94.99	1
EO	6002587	NELLISEN	15.00	
EO	7082509	REED	5.00	
EO	6208657	SMYTH	32.00	
control line 1 →			52.00	3
final line 1 →			146.99	4

Figure 318 TALLY Example 1

Taking the same example, TALLY calculates the overall average of BALANCE per account at ON FINAL time.

```

LIST ACCOUNT TRANS NAME BALANCE
ON CHANGE IN ACCOUNT
LIST SUM BALANCE TALLY WITH 2 BEFORE AND 2 AFTER
ON FINAL
AVERAGE=SUM BALANCE/TALLY ROUNDED ←
LIST '***AVERAGE BALANCE' AT ACCOUNT AVERAGE AT NAME ←
SUM BALANCE TALLY WITH 2 BEFORE

```

ACCOUNT	TRANS	NAME	BALANCE	TALLY
BO	8006547	TOWNSON	94.99	
			94.99	1
EO	6002587	NELLISEN	15.00	
EO	7082509	REED	5.00	
EO	6208657	SMYTH	32.00	
			52.00	3
***AVERAGE BALANCE		36.75	146.99	4

Figure 319 TALLY Example 2

In the above statements, VISION:Results is alerted to print a grand total line. Contained within the final line is a figure representing the sum of all balances (SUM BALANCE) divided by the number of accounts (TALLY) reported.

$146.99/4 = 36.747$
 $= 36.75 \text{ ROUNDED}$

The following example demonstrates how to use TALLYn:

```

CONTROL ACCOUNT DIVISION
ON CHANGE IN ACCOUNT
LIST DIVISION PRIOR ACCOUNT TALLY1
ON CHANGE IN DIVISION
LIST PRIOR DIVISION ACCOUNT TALLY2

```

Figure 320 TALLYn Example

This creates the following report lines for the control break LIST statements:

DIVISION	ACCOUNT	BALANCE	TALLY1	TALLY2
1	012345	500.00		
	012345	110.99		
	012345		2	
	023345	222.55		
	023345	9.99		
	023345		2	
	456789	4.00		
	456789	10000.00		
	456789	.00		
	456789		3	
1	456789			7
2	098765	9999.99		

TALLY is one of the few reserved words in VISION:Results that can have its output print size and edit format overridden in a WORKAREA definition. TALLY is defined by VISION:Results as a 16-byte packed field with nine decimal positions, with a default edit code of Z and a column heading default of TALLY. For printing purposes, TALLY defaults to a maximum size of 10 (9999999999).

Because the TALLY print size takes up considerable space on the print line, it is sometimes necessary to use overrides to shorten the TALLY print size, change the TALLY edit code, and/or give the TALLY a unique column heading.

For example, to give TALLY a unique column heading and change its edit code to E (comma insertion), you can supply the following:

```
WORKAREA  
TALLY E (COUNT)
```

The maximum print size is 9,999,999,999 (commas included, the print size is now 13).

If you want to limit the number of print digits (size), code the following:

```
WORKAREA  
TALLY E 4 (TOTAL'ACCOUNTS')
```

where 4 (number of significant digits) and an E edit code would result in a maximum print size of 9,999.

Column Heading

The literal data entered in the column heading position of the parameter appears on the report as the column heading for the field specified by the data name. The column heading data must be enclosed in parentheses.

Column headings specified in the LIST statement override those specified in the field definition. The maximum length of any column heading is 30 positions.

If no column heading is entered in the LIST statement or field definition, the default is the data name of the field.

```
CONTROL DEPARTMENT  
LIST SUPPRESS DEPARTMENT (DEPT) EMPNAME (EMPLOYEE NAME) NETPAY (NET PAY)  
ON CHANGE IN DEPARTMENT  
LIST SUM NETPAY WITH 2 BEFORE AND 2 AFTER
```

DEPT	EMPLOYEE NAME	NET PAY
1	JOHN JONES	865.98
	MARY POLLACK	987.88
	STEVEN SMITH	678.98
		2532.84

Figure 321 Column Heading Example 1 (Page 1 of 2)

2	ALBERT AMES	587.23
	JOHN CARLS	989.90
		1577.13
3	ANITA BAIRD	667.77
	FRANK KOCH	348.55
		1016.32

Figure 321 Column Heading Example 1 (Page 2 of 2)

To stack column headings, you use an apostrophe before, between, or after the headings. The maximum length, including the apostrophes, for any column heading is 30 characters. A maximum of nine column headings can be stacked.

For example:

```
CONTROL DEPARTMENT
LIST SUPPRESS DEPARTMENT (D'E'P'T) EMPNAME (EMPLOYEE'NAME) NETPAY ('NET'PAY)
ON CHANGE IN DEPARTMENT
LIST SUM NETPAY WITH 2 BEFORE AND 2 AFTER
```

D E P T	EMPLOYEE NAME	NET PAY
1	JOHN JONES	865.98
	MARY POLLACK	987.88
	STEVEN SMITH	678.98
		2532.84
2	ALBERT AMES	587.23
	JOHN CARLS	989.90
		1577.13
3	ANITA BAIRD	667.77
	FRANK KOCH	348.55
		1016.32

Figure 322 Column Heading Example 2

To obtain a blank column heading that overrides a predefined column heading definition or the default of the data name, use parentheses () with 0 to 30 blanks between the parentheses.

For example:

```
CONTROL DEPARTMENT
LIST SUPPRESS DEPARTMENT ( ) EMPNAME (EMPLOYEE'NAME) NETPAY ( )
ON CHANGE IN DEPARTMENT
LIST SUM NETPAY WITH 2 BEFORE AND 2 AFTER
```

	EMPLOYEE NAME	
1	JOHN JONES	865.98
	MARY POLLACK	987.88
	STEVEN SMITH	678.98
		2532.84

Figure 323 Column Heading Example 3 (Page 1 of 2)

```

2          ALBERT AMES      587.23
          JOHN CARLS        989.90
                                     1577.13

3          ANITA BAIRD      667.77
          FRANK KOCH        348.55
                                     1016.32

```

Figure 323 Column Heading Example 3 (Page 2 of 2)

In summary, column headings are used as follows:

- Column headings on the LIST statement take priority.
- The column heading is taken from the field definition if none is specified on the LIST statement.
- The data name becomes the column heading if none is specified in the field definition.
- If you do not want a column heading, specify a null heading, that is, ().
- For any field aligned (AT dataname or AT dataname +n) to another, either by you or automatically, you get a column heading only if it is defined in your LIST statement. VISION:Results does not default to the data name.
- Column headings are not available for fixed print position reporting. However, fixed print column headings can be specified with TITLE statements. See [FIXED on page 372](#).

Print Line Spacing

```

WITH {n} BEFORE AND {n} AFTER
    {EJECT}

```

Figure 324 Print Line Spacing

This statement controls the line spacing before and after the printing of each detail line.

WITH n BEFORE and WITH EJECT BEFORE

The default spacing before printing is WITH 0 BEFORE. You can specify from zero to nine lines of spacing before detail line 1 or control line 1 is printed.

```

ON CHANGE IN ACCOUNT
LIST ACCOUNT SUM BALANCE WITH 2 BEFORE

```

Figure 325 WITH n BEFORE Spacing

Page ejection and a skip to channel 1 (top of next page) can be specified in detail line 1, control line 1, or final line 1.

```
ON CHANGE IN ACCOUNT
  LIST ACCOUNT SUM BALANCE WITH EJECT BEFORE
```

Figure 326 WITH EJECT BEFORE Spacing

Spacing BEFORE is valid only for the first LIST statement at any level.

WITH n AFTER and WITH EJECT AFTER

The default spacing for WITH n AFTER is 1. You can specify from zero to nine lines of spacing after print detail line 1, control line 1, or final line 1.

```
ON CHANGE IN ACCOUNT
  LIST ACCOUNT SUM BALANCE WITH 3 AFTER
```

Figure 327 WITH n AFTER Spacing

You can specify from 0 to 99 lines of spacing on all subsequent detail, control, and final lines. If you code WITH 0 AFTER on two or more consecutive detail, control, or final LIST statements, the result is overstriking (overprinting). You can use this feature to make a field appear in boldface when printed.

```
IF BALANCE GT 10000
  LIST ACCT BALANCE WITH 0 AFTER
  LIST ACCT BALANCE WITH 0 AFTER
  LIST ACCT BALANCE
```

Figure 328 WITH n AFTER Spacing on Defined Lines

Both the ACCOUNT field and the BALANCE field are overprinted on the report.

Page ejection, a skip to channel 1, is specified in the AFTER form. For example:

```
ON CHANGE IN ACCOUNT
  LIST ACCOUNT SUM BALANCE WITH EJECT AFTER
```

Figure 329 WITH EJECT AFTER Spacing

The LIST statement allows the use of EJECT without BEFORE or AFTER.

Up to 99 LIST statements can be used at any one level. The first level is the detail level and is comprised of the code up to the first ON CHANGE IN or first ON FINAL statement. Subsequent levels are between each ON CHANGE IN or between the last ON CHANGE IN or ON FINAL and the end of the program.

Alignment of Data Names and Literals at Detail Time (Multi-line)

With the VISION:Results automatic composition feature, the data names for second and subsequent detail lines (up to 99) are automatically lined up column-for-column with the data names appearing in the first detail line. Additional columns are created to the right of the last aligned column to accommodate additional data names. If you want to align the fields differently, use the dataname AT dataname2 option.

You can also specify dataname AT dataname2 +n (alignment plus a displacement to the right). For example, LIST ADDRESSL1 AT NAME + 5.

The following rules govern alignment for data names or literals specified on detail lines 2 through 99 for which AT has not been specified:

- VISION:Results decimal point aligns the data name (if binary, numeric, or packed) under the data to which it is aligning.
- If the data name field size is greater than the larger of the column heading or the field to which it is being aligned, VISION:Results left-aligns the data name with the field/column heading to which it is aligning.
- If the data name field size is smaller than the data name to which it is aligning, VISION:Results centers the data name within the column.

For example:

```
LIST VENDNO ITEM CODE DESCRIPT      ← detail line 1
LIST LOCATION ONHAND UNITS ONORDER  ← detail line 2
    WITH 2 AFTER
```

VENDOR	ITEM	CODE	DESCRIPT
JOHN SMYTH SAN FRANCISCO	SHOES 20	SF PAIRS	ALLIGATOR/LIZARD 40
PAUL DOE LOS ANGELES	STRING 1000	SS PAIRS	BROWN 200

Figure 330 Data Name Field Size is Smaller

LOCATION is aligned (left-aligned) under VENDNO, and so on.

For example, to shift LOCATION to align under ITEM:

```
LIST VENDNO ITEM CODE DESCRIPT      ← detail line 1
LIST LOCATION AT ITEM ONHAND UNITS ONORDER  ← detail line 2
    WITH 2 AFTER
```

VENDOR	ITEM	CODE	DESCRIPTION	ON ORDER
JOHN SMYTH SAN FRANCISCO	SHOES 20	SF PAIRS	ALLIGATOR/LIZARD PAIRS	40
PAUL DOE LOS ANGELES	STRING 1000	SS PAIRS	BROWN PAIRS	200

Figure 331 Shift LOCATION

Because LOCATION was aligned AT ITEM, ONHAND shifts to align under CODE and UNITS shifts to align under DESCRIPTION.

ONORDER creates its own column with the appropriate heading.

If you specify an alignment for a data name or literal, all subsequent data names or literals in the second or subsequent line are lined up to the right of the alignment.

Taking the same example a third time, displace ONORDER 25 positions beyond DESCRIPT.

```
LIST VENDNO ITEM CODE DESCRIPT
LIST LOCATION ONHAND UNITS ONORDER AT DESCRIPTION+25
  WITH 2 AFTER
```

VENDOR	ITEM	CODE	DESCRIPTION	
JOHN SMYTH SAN FRANCISCO	SHOES 20	SF PAIRS	ALLIGATOR/LIZARD	40
PAUL DOE LOS ANGELES	STRING 1000	SS PAIRS	BROWN	200

Figure 332 Displace ONORDER 25 Positions

No column heading was produced for ONORDER. To do so, you need to specify the column heading in the LIST statement. For example, to create the necessary column heading for ONORDER:

```
LIST LOCATION ONHAND UNITS ONORDER AT DESCRIPTION+25 (ON'ORDER)
```

Figure 333 Column Heading for ONORDER

Alignment of Data Names and Literals at ON CHANGE IN and ON FINAL Time

With the VISION:Results automatic composition feature, the data names for second and subsequent control lines are automatically lined up column-for-column with the data names appearing in the first control or final line. This occurs unless the data name appears in a detail LIST, in which case it aligns under that column.

The rules for alignment are the same as those presented in the previous section on alignment of data names and literals at detail time.

This example creates a two-line summary report of all accounts:

```
CONTROL ACCOUNT
ON FINAL
  LIST SUM (BALANCE INSTLBAL INSTPAY BALPART)
    WITH 2 BEFORE AND 2 AFTER
LIST TALLY 'ACCOUNTS'
```

Figure 334 Two-Line Summary Report

The resulting report looks like this:

BALANCE	INSTALL BALANCE	INSTALL PAYMENT	BALANCE PARTIAL
6,892.92	5,311.00	1,581.92	322.13
200	ACCOUNTS		

Figure 335 Two-Line Summary Report Results

In accordance with the rules of alignment, TALLY is automatically aligned under the column heading BALANCE while the literal ACCOUNTS is aligned under INSTALL BALANCE.

Taking the same example, a summary of (control line 1) is created and the average for each account (control line 2) is calculated.

```

CONTROL ACCOUNT
ON CHANGE IN ACCOUNT
  AVG1=SUM BALANCE/TALLY
  AVG2=SUM INSTLBAL/TALLY
  AVG3=SUM INSTLBAL/TALLY
  AVG4=SUM BALPART/TALLY
→ LIST ACCOUNT SUM (BALANCE INSTLBAL INSTPAY BALPART)
  WITH 2 BEFORE AND 2 AFTER
→ LIST AVG1 AVG2 AVG3 AVG4 WITH 2 AFTER
ON FINAL
  LIST SUM (BALANCE INSTLBAL INSTPAY BALPART)
  WITH 2 BEFORE AND 2 AFTER
LIST TALLY 'ACCOUNTS'

```

ACCOUNT	BALANCE	INSTALL BALANCE	INSTALL PAYMENT	BALANCE PARTIAL
BO	119.22	28.00	101.02	98.10
119.22	28.00	101.02	98.10	← control line 2
EO	201.22	198.44	102.86	7.44
100.66	99.22	51.43	3.72	← control line 2
		.		
		.		
		.		
	6,892.92	5,311.00	1,581.92	322.13
	200	ACCOUNTS		← final line 2

Figure 336 Summary of (Control Line 1) is Created

AVG1 (control line 2), which is the average of BALANCE, aligned itself under ACCOUNT and AVG2 under BALANCE, and so on. Remember, the rules of alignment state that data names align column-for-column with those specified in control line 1. This is also apparent on final line 2. TALLY aligned itself under BALANCE, the first data name specified at final line 1.

If you want AVG1 to align under BALANCE, AVG2 under INSTLBAL and so on, there are two possible ways to accomplish this and get the proper alignment:

Specify a literal to print first, forcing AVG1 under BALANCE, and so on:

```
LIST 'AVERAGE' AVG1 AVG2 AVG3 AVG4
```

Figure 337 Forcing AVG1 Under BALANCE

Or, use a literal of spaces:

```
LIST ' ' AVG1 AVG2 AVG3 AVG4
```

Figure 338 Literal of Spaces

Using the same example, detail print all records (suppressing ACCOUNT), print TALLY at control line 1, print AVG1 aligned under BALANCE on control line 2, and create a third control line with the literal CORRECT _____.

```
CONTROL ACCOUNT
→ LIST SUPPRESS ACCOUNT BALANCE INSTLBAL INSTPAY BALPART
ON CHANGE IN ACCOUNT
  AVG1=SUM BALANCE/TALLY
  AVG2=SUM INSTLBAL/TALLY
  AVG3=SUM INSTLBAL/TALLY
  AVG4=SUM BALPART/TALLY
→ LIST ACCOUNT SUM (BALANCE INSTLBAL INSTPAY BALPART) TALLY
  WITH 2 BEFORE AND 2 AFTER
→ LIST ' ' AVG1 AVG2 AVG3 AVG4 WITH 2 AFTER
→ LIST 'CORRECT _____' WITH 3 AFTER
ON FINAL
  LIST SUM (BALANCE INSTLBAL INSTPAY BALPART)
  WITH 2 BEFORE AND 2 AFTER
  LIST TALLY 'ACCOUNTS'
```

Figure 339 Alignment Example 1

The resulting report looks like this:

ACCOUNT	BALANCE	INSTALL BALANCE	INSTALL PAYMENT	BALANCE PARTIAL	TALLY
BO	119.22	28.00	101.02	98.10	
BO	119.22	28.00	101.02	98.10	1
	119.22	28.00	101.02	98.10	
CORRECT _____					← control line 3
EO	100.22	108.22	80.86	.00	
	101.00	90.22	22.00	7.44	
EO	201.22	198.44	102.86	7.44	2
	100.66	99.22	51.43	3.72	
CORRECT _____					← control line 3
		.			
		.			
		.			
	6,892.92	5,311.00	1,581.92	322.13	
		ACCOUNTS			200

Figure 340 Alignment Results 1

Fixed Print Position Reporting

The LIST statement can be used to indicate the contents of a report line and to specify the print locations for the placement of each field. This form of LIST is used when you want to format your own report. You cannot mix fixed print position LIST statements with the automatic positioning LIST statement.

The format of this form of the LIST statement is:

```
LIST dataname1 ![datanamen]!..  
      [WITH {n} BEFORE AND {n} AFTER]  
      {EJECT} {EJECT}
```

Figure 341 Print Locations

The keyword LIST begins each statement. The remaining elements in the LIST statement are positional. They must appear in the order shown. Fields appear on the report in the location you specify. When using this form of LIST, column headings are not printed or allowed.

Data Name

Each data name in the LIST statement identifies a field to be printed. It also can specify operations to be performed on the field. See [LIST Statement on page 378](#).

Each data name is positional and has one of the following formats:

Format 1:

```
[qualifier] dataname AT nnn
```

Figure 342 Data Name Position Format 1

Format 2:

```
literal AT nnn
```

Figure 343 Data Name Position Format 2

Format 3:

```
TALLY AT nnn
```

Figure 344 Data Name Position Format 3

Each element in this form of the LIST statement is identical to that in the previous format except for print location (nnn). The AT data name alignment feature is not supported under this format. The EDALIGN DYLINSTL parameter will not be in effect under this format. For descriptions of qualifier, data name, literal, WITH n BEFORE, and WITH n AFTER, see [LIST Statement on page 378](#). Column headings are not supported with this format and, if required, you must define the column headings utilizing the FIXED format of the TITLE (T1-T9) statement. See [Title or Footing Modification Statement on page 374](#).

nnn (Print Location)

When using this form of the LIST statement, you must enter a number indicating the beginning print location on the report for each field. Therefore, with this form of LIST, the data name for the field and the print location must be specified in each parameter. After the data name, the keyword AT must precede the print location number.

For example:

```
LIST DEPARTMENT AT 1 EMPNAME AT 30 NETPAY AT 60
```

Figure 345 nnn Print Location

In the above example of a fixed position LIST, the value of DEPARTMENT is printed at print position 1, EMPNAME is printed at position 30, and NETPAY is printed at position 60 of the report line.

The last position of any print line cannot be used.

For a complete example of fixed position printing, see [Report in Fixed Print Position on page 420](#).

CONTROL or SUBTOTAL Statement

The CONTROL or SUBTOTAL statement defines the control breaks wanted (SUBTOTAL is a synonym for CONTROL). This is done by specifying the data name or data names of the field definitions involved. From one to six control break levels can be specified in the CONTROL or SUBTOTAL statement. You must define all of the data names before specifying them in the CONTROL or SUBTOTAL statement.

The format of the CONTROL statement is:

```
CONTROL control break 1 ...control break 6
```

Figure 346 CONTROL Statement Format

The format of the SUBTOTAL statement is:

```
SUBTOTAL [BY] control break1 ...control break 6
```

Figure 347 SUBTOTAL Statement Format

The CONTROL or SUBTOTAL statement can be placed anywhere in the procedure logic of your program, but it must always precede the first ON CHANGE IN statement.

A control break is said to occur when VISION:Results recognizes a change in the value of any of the specified fields. For example:

```
CONTROL ACCOUNT  
or  
SUBTOTAL ACCOUNT
```

Figure 348 CONTROL or SUBTOTAL Statement Format

Whenever a change in the value of the field called ACCOUNT occurs (a control break occurs), the ON CHANGE IN ACCOUNT logic is executed. However, it works in unison with the CONTROL or SUBTOTAL statement and describes what is to be done when a specific control break occurs. The ON CHANGE IN statement is described in [ON CHANGE IN on page 409](#).

For example:

```
CONTROL DEPT DIV  
:  
:  
ON CHANGE IN DEPT  
  LIST SUM NETPAY WITH 2 BEFORE AND 2 AFTER  
ON CHANGE IN DIV  
  LIST SUM OVERTIME SUM NETPAY WITH 2 BEFORE
```

Figure 349 CONTROL Statement Example

Whenever a change in the value of DIV occurs, the lower-level control break logic for DEPT (ON CHANGE IN) is executed first before the higher-level DIV (ON CHANGE IN) is executed.

Therefore, the first data name in the CONTROL or SUBTOTAL statement signals the first control break (lowest or most minor), the second signals the second, and so on. A higher-level control break always causes all lower breaks to occur.

SUBTOTAL and SUBTOTAL BY are synonyms for CONTROL.

In all VISION:Results programs, there is a final, grand total level that occurs as part of the normal VISION:Results cycle. You do not need to specify the field on which the final level is to occur. See [ON FINAL on page 410](#).

ON CHANGE IN

The ON CHANGE IN tells VISION:Results the logic to be executed or lines to be printed when a change in value is detected. The format is:

```
ON CHANGE IN dataname
```

Figure 350 ON CHANGE IN

Data Name

The data name specified must have been defined in a CONTROL or SUBTOTAL statement and the CONTROL or SUBTOTAL statement must precede the first ON CHANGE IN statement. For example:

```
CONTROL ACCOUNT
.
.
LIST SUPPRESS ACCOUNT TRANS NAME BALANCE
ON CHANGE IN ACCOUNT
LIST SUM BALANCE WITH 2 BEFORE AND 2 AFTER
```

Figure 351 ON CHANGE IN Dataname

The ACCOUNT field is checked for a change in value before listing the detail line. If there is a change in the value of ACCOUNT, the logic coded after the ON CHANGE IN ACCOUNT is executed. The detail line is then printed.

The keyword IN is a noise word and is not required in the ON CHANGE statement. The following statements have the same meaning:

```
ON CHANGE IN ACCOUNT
ON CHANGE ACCOUNT
```

Figure 352 ON CHANGE IN Without Keyword IN

If a LIST statement is specified as part of the logic following the ON CHANGE, it must be the last statement specified, with the following exceptions:

- Another LIST
- Tn or Tn+n
- ACCEPT
- Comments (* or ;)

```
ON CHANGE IN ACCOUNT
MOVE ACCOUNT TO WACCT
LIST SUM BALANCE WITH 2 BEFORE
```

Figure 353Comments

Additionally, you can execute many other commands to build and write summary records, compute formulas, or test the total of fields (SUM).

```
ON CHANGE IN ACCOUNT
  MOVE SUM BALANCE TO WBALANCE
IF SUM BALANCE GE 300
  MOVE '***' TO REMARKS ENDIF
AVG=SUM BALANCE/TALLY
LIST SUM BALANCE .....
```

Figure 354 Other Commands

The ON CHANGE IN need not be supplied in the order of the control break levels as specified in the CONTROL statement. However, the CONTROL statement must always precede the first ON CHANGE IN.

```
CONTROL ACCOUNT
ON CHANGE IN ACCOUNT
  LIST SUM BALANCE

CONTROL TRANS ACCOUNT
ON CHANGE IN ACCOUNT
  LIST SUM BALANCE
ON CHANGE IN TRANS
  LIST SUM BALANCE
```

Figure 355 ON CHANGE IN Not Supplied in Order

If you want a control break but do not want anything to be printed, use the following statements to create a blank line between control breaks.

```
ON CHANGE IN ACCOUNT
  LIST ' ' WITH 1 AFTER
```

Figure 356 Control Break

To eject the page between control breaks:

```
ON CHANGE IN ACCOUNT
  LIST ' ' WITH EJECT AFTER
```

Figure 357 Eject the Page

ON FINAL

The ON FINAL statement defines the final, grand total logic, specified by you, that is automatically executed by the system when end-of-processing occurs. When all records have been reported on, you might want to print overall totals or summarize information on your report.

The format is:

```
ON FINAL
```

Figure 358 ON FINAL Statement

This statement precedes the logic you want performed before the report is finished.

Nothing needs to be coded in the CONTROL statement to indicate the final, grand total level. This event is not considered a control break level.

The ON FINAL statement is required only if you want additional wrap-up logic to be executed before the report and job come to a close.

The rules governing the ON CHANGE IN also apply to ON FINAL.

For example:

```
ON FINAL  
  LIST 'GRAND TOTAL' AT NAME SUM NET
```

Figure 359 ON FINAL Statement Example

The grand total of NET is to be printed, preceded by the literal GRAND TOTAL printed underneath the NAME field on the report.

In a VISION:Results program, you cannot fall into ON CHANGE IN or ON FINAL logic. VISION:Results assumes that an ACCEPT occurs prior to any ON CHANGE IN or ON FINAL statement.

Report Examples

Example 1 Detail Report, One Line

Produce a detail report of all vendors (number, name, address, city, state, zip code) on an Accounts Payable file. Generate the report with no titles, no date, and no page numbering, but include column headings.

```
FILE VENDOREL FB 100 1000
RECTYPE 2  VENDORNO 5  VENDNAME 20  TITLE 20
ADDRESS 20  CITY 20  STATE 2  ZIP 5
LIST VENDORNO (VENDOR NUMBER) VENDNAME (VENDOR NAME)
TITLE (ATTN OF TITLE) ADDRESS CITY STATE ZIP
```

VENDOR NUMBER	VENDOR NAME	ATTN OF TITLE	ADDRESS	CITY	STATE	ZIP
00012	COUNTY ASSESSOR	CITY CLERK	120 W. FIRST ST	LOS ANGELES	CA	90014
00014	COMPUTER ASSOCIATES		16255 VENTURA BLVD	ENCINO	CA	91346
00020	FREE PRESS		512 W. 6TH STREET	LOS ANGELES	CA	90016
00032	ITTE BITTE MONOPOLY		3455 WILSHIRE BLVD	LOS ANGELES	CA	90018
00161	DEMINGTON OFC MACH		6873 LAKE VIEW DRIVE	THOUSAND ISLANDS	CA	91560
00263	DURROUGHES CORP		10331 ZELZAH AVE	EASTFORD	CA	91306
00268	FLUE MOSS INSURANCE		440 N JIMINEZ BLVD	LOS ANGELES	CA	90036
00271	COMMON ELECTRIC CORP		731 LA CIENEGA	LOS ANGELES	CA	90020
00275	SPECIFIC TELE CO		8118 SAN VINCENT AVE	LOS ANGELES	CA	90015
00305	DARWIN REALTY		12016 NEWCASTLE AVE	ENCINO	CA	91436
00323	APPO OFFICE CLEAN		457 HASKELL AVE	SEPULVEDA	CA	91324
00340	KAPERMATE CO		1000 LEXINGTON LN	NEW YORK	NY	10017
00358	ITS TOO TOO CORP	ACCTS RCV	24 HATTERAS RD	LITTLE ROCK	PA	82306
00390	UP-AND-AWAY TRAVEL	PHYLLIS	7260 DESOTO	HOLLYWOOD	CA	91604
00415	RONS ROBBERY RUIN CO		601 ETTWANDA	CHATSORTH	CA	91311
00418	GOLD STAR OFC EQUIP		153 HESBY AVE	AGOURA	CA	91301
00429	SCROLL PAPER SPLY		53401 CHANDLER	SYLMAR	CA	91405
00436	WOMANSTRONG AGENCY	GLORIA	3255 MIDVALE	WESTWOOD	CA	90024
00500	QUILL PRINTERS		6447 SATSUMA RD	TUJUNGA	CA	91042

Figure 360 [Example 1](#) Detail Report, One Line

Example 2 Summary Report, One Final Line

Produce a summary report on depreciated fixed assets. Print a summary line with titles and column headings, selecting only those items belonging to category 260.

```
FILE FIXASSET FB 100 1000
COST 6 40 PD 2 E 7.2 (COST BASIS) CATEGORY 4 17
DEBPRIOR 6 46 PD 2 E 7.2 (DEPRECIATION PRIOR)
DEBAPTER 6 52 PD 2 E 7.2 (DEPRECIATION AFTER)
GAINLOSS 6 70 PD 2 E 7.2 (TOTAL GAIN/LOSS)
RECOGLOSS 6 76 PD 2 E 7.2 (TOTAL RECOG. LOSS)
ORDGAIN 6 82 PD 2 E 7.2 (TOTAL ORD. GAIN)
OTHER 6 88 PD 2 E 7.2 (TOTAL OTHER GAIN)
WORKAREA
CTR 4 PD Z (ITEMS) VALUE 1
IF CATEGORY EQ '260 ' NEXT ELSE REJECT ENDIF
```

Figure 361 [Example 2](#) Summary Report, One Final Line (Page 1 of 2)

ON FINAL
 LIST SUM CTR SUM COST SUM DEPRIOR SUM DEPARTER
 SUM GAINLOSS
 SUM RECOGLOSS SUM ORDGAIN SUM OTHER
 T1 'DEPRECIATION RECAPTURE' T2 'FIXED ASSETS'
 T3 'BY CATEGORY 260' WITH 2 AFTER

DEPRECIATION RECAPTURE
 FIXED ASSETS
 BY CATEGORY 260

ITEMS	COST BASIS	DEPRECIATION PRIOR	DEPRECIATION AFTER	TOTAL GAIN/LOSS	TOTAL RECOG. LOSS	TOTAL ORD. GAIN	TOTAL OTHER GAIN
14	56,034.97	60,201.80	68,718.44	61,666.81	68,712.78	78,511.66	50,187.12

Figure 361 [Example 2](#) Summary Report, One Final Line (Page 2 of 2)**Example 3 Detail Report, One Detail Line and One Final Line**

Generate a personnel detail report with a total count of all employees. Include their department number, employee number, name (first second last), and name again (last first second). Also, include their extension and home telephone number. Number all pages and include today's date with the appropriate titles and column headings.

FILE EMPYFILE FB 80 800
 DEPT 3 1 NUMBER 5 4 (EMPLOYEE'NUMBER)
 NAME 32 9 (EMPLOYEE NAME) ALSTNM 15 26
 AFIRST 17 9 EXT 4 41 (WORK EXT) LOCATION 10 45
 HOMEPEH 12 55 (HOME NUMBER)
 WORKAREA
 FULLNAME 32 1 (EMPLOYEE NAME) LASTNAME 15 1
 FIRSTMID 17 16 TALLY Z 4
 MOVE ALSTNM TO LASTNAME
 MOVE AFIRST TO FIRSTMID
 LIST DEPT NUMBER NAME EXT LOCATION FULLNAME HOMEPEH
 ON FINAL
 LIST TALLY (COUNT) WITH 1 BEFORE
 T1
 T1+50 'COMPUTER ASSOCIATES' T1+116 'DATE' T1+125 DYLDATE
 T2 WITH 2 AFTER
 T2+56 'PHONE LIST' T2+116 'PAGE' T2+128 DYLPAGE4

COMPUTER ASSOCIATES PHONE LIST							DATE	01/15/01
							PAGE	1
DEPT	EMPLOYEE NUMBER	EMPLOYEE NAME	WORK EXT	LOCATION	EMPLOYEE NAME	HOME NUMBER	COUNT	
011	48205	LORAC	B. CARROLL	3110	CLERKTYP CARROLL	LORAC	B. 213 877-1235	
010	46044	DUDLEY	G. DORIGHT	8271	AUDIT SEC DORIGHT	DUDLEY	G. 213 241-8582	
011	46050	FANNIE	FAIRWEATHER	2481	CLERKTYP FAIRWEATHER	FANNIE	213 762-7360	
014	48920	GINGER	A. FIZZ	1593	SECRTY DP FIZZ	GINGER	A. 213 763-3240	
014	48844	GUSSIE	GASPARD	4180	KEYPUNCH GASPARD	GUSSIE	213 479-8190	
011	47020	SPEEDBALL	GONZALES	2482	MAIL ROOM GONZALES	SPEEDBALL	213 685-3481	
050	49230	SEEMORE	HORIZONS	8641	MAINT HORIZONS	SEEMORE	714 347-6629	
030	47030	BERNIE	MILKTOAST	3012	COMPUTR RM MILKTOAST	BERNIE	213 782-3555	
001	49021	MYRTLE	MUMFORD	2222	SR VP MUMFORD	MYRTLE	805 564-3111	
030	48009	SEL	NOSLIW	6281	COMPUTR RM NOSLIW	SEL	213 987-3222	
001	48325	ODDFORD	COGLEHOFF	4444	VICE PRES COGLEHOFF	ODDFORD	213 341-6791	
011	48605	PETER	POOPERDINK	2482	MAIL ROOM POOPERDINK	PETER	213 444-0484	
030	46055	ROBERT	T. RABBIT	3012	COMPUTR RM RABBIT	ROBERT	T. 213 832-3211	
025	49015	OSCAR	RAIDER	3531	TECH WRTR RAIDER	OSCAR	805 884-0982	
001	47012	WOLFHOUND	ROADRUNNER	3499	EXEC JR ROADRUNNER	WOLFHOUND	805 242-0390	
010	49105	TOTO	R. ROTO	5415	AUDITOR ROTO	TOTO	R. 213 342-8741	
001	47930	TREBOR	S. ROBERT	3422	EXEC SR ROBERT	TREBOR	S. 714 382-7211	
050	49244	SUDSY	B. SUGGINS	7498	MAINT SUGGINS	SUDSY	B. 805 777-0150	
014	49125	THOR	VIKING	3675	KEYPUNCH VIKING	THOR	213 845-4886	
014	47525	STANLEY	YELNATS	6211	DP MGR YELNATS	STANLEY	213 655-3297	
030	47115	GODFREY	Q. ZANE	7821	OPER MGR ZANE	GODFREY	Q. 213 793-8763	
011	49500	ZAC	Z. ZOOK	5862	CLERKTYP ZOOK	ZAC	Z. 213 360-6258	

22

Figure 362 [Example 3](#) Detail Report, One Detail Line and One Final Line

Example 4 Report with One Detail Line, Two Control Lines, and One Final Line

Create a job accounting report to cover all types of abnormal job terminations encountered. The report should contain the programmer's name, job name, program name, abend codes, and so on. Control breaks should occur on a change in programmer's name and a change in system abend code.

```
FILE JOBACCT FB 54 5400
SYSCODE 1  ABENDCODE 3  (ABEND'CODE)
JOBNAME 8  (JOB'NAME)
PGMNAME 8  (PROGRAM'NAME)  STEPNAME 8  (STEP'NAME)
STEPDATE 6  NU D (STEP'DATE)
TIME 4  NU 2  (STEP'TIME'[HH.HH])
ACCINFO 6  (ACCTG.'INFO.)  PGMNAME 8  (PROGRAMMER)

CONTROL PGMNAME SYSCODE
LIST SUPPRESS PGMNAME JOBNAME PGMNAME SUPPRESS
      ABENDCODE STEPNAME
SUPPRESS STEPDATE TIME SUPPRESS ACCINFO
ON CHANGE IN PGMNAME
LIST TALLY (ABEND'COUNT) WITH 1 BEFORE AND 2 AFTER
ON CHANGE IN SYSCODE
LIST TALLY (ABEND'COUNT) 'TOTAL ABENDS' AT JOBNAME+5
      WITH 1 BEFORE AND 2 AFTER
ON FINAL
LIST TALLY (ABEND'COUNT) 'TOTAL ABENDS' AT JOBNAME+5
      WITH 1 BEFORE AND 2 AFTER

T1 'JOB ACCOUNTING REPORT'
T1+1 'COMPUTER ASSOCIATES'
T1+108 'PAGE'
T1+114 DYLPAGE4
T2 'ABNORMAL TERMINATIONS' WITH 2 AFTER
T2+1 'RPT.DATE'
T2+12 DYLDATE
```

Figure 363 [Example 4](#) Report with One Detail Line, Two Control Lines, and One Final Line

COMPUTER ASSOCIATES		JOB ACCOUNTING REPORT						PAGE	
RPT.DATE 01/15/01		ABNORMAL TERMINATIONS						1	
PROGRAMMER	JOB NAME	PROGRAM NAME	ABEND CODE	STEP NAME	STEP DATE	STEP TIME [HH.HH]	ACCTG. INFO.	ABEND COUNT	
CORRIGAN	CRASH	AP200	D23	AGAIN1	01/15/01	00.30	201800		
	WRONGWAY	AP300	E0	AGAIN2	01/16/01	00.08	201801		
	TIMEOUT	AP300	B37	NOTAGAIN	01/15/01	01.31	201802		3
DUOTANG	NOSEDIVE	AR570	222	S001	01/15/01	00.42	300201		
	WHODUNIT	AR590	322	S003		00.15	300202		2
SMITTEN	OHBOY	DYL250	E37	STEP5	01/15/01	00.13	301800		1
SUNK	BLOWNIT	MC310	062	STEPOUT1	01/15/01	00.07	301800		
	ATHEEEEE	MC330	0A1	STEPOUT3		01.13			
	THEBOMB	MC3000P	102	STEPOUT4	01/28/01	00.22	302300		
	ELEWIT	MC110	122	TOOMANY		00.35			
	GOODNESS	MF201	13F	ONCEMORE		00.25	301800		5
TOTAL ABENDS									11

Figure 364 [Example 4](#) Report with One Detail Line, Two Control Lines, and One Final Line Results (Page 1 of 2)

SEAWER	SLADE	AP200	499	STONE1	01/15/01	00.30	201800	3
	SLADE	AP302	500	STONE4	01/16/01	00.08		
	TIMER	DYL280	476	SOFEW	01/15/01	01.31	201802	
HUNTER	DOWNONE	AR570	222	5001	01/15/01	00.42	300201	2
	DOWNTWO	AR590	322	S003		00.15	300202	
DOOM	BLOWNIT	MC310	562	STEP0UT1	01/15/01	00.07	301800	4
	ATREEEE	MC330	495	STEP0UT3		01.13		
	THEEOM8	MC300OPT	502	STEP0UT4	01/16/01	00.22	302300	
	BLEWITT	MC110	522	TOOMPANY		00.35		
TOTAL ABENDS								9
TOTAL ABENDS								20

Figure 364 [Example 4](#) Report with One Detail Line, Two Control Lines, and One Final Line Results (Page 2 of 2)

Example 5 Report with Five Detail Lines

Generate an itemized list of all products purchased and in inventory for vendor 0021.

```

FILE JOBACCT VB 200 LENGTH RECLLENGTH
NUMBER 4 (VEN'NO.) ITEMNO 5 (ITEM) COMCODE 2 (COMM'CODE)
NOLOCTN 2 NU (NO.OF'LOC.) DESCN 35 ('DESCRIPTION)
UNITMEAS 2 ('U/M) WEIGHT 3 ('WGT)
COST 4 PD E 3.2 ('COST)
UNITHAND 4 PD E 5 (TOTAL'ON HAND)
ONORDER 4 PD E 5 (ON'ORDER)
AVGUSE 3 PD E 5 (AVG'USAGE) AVGMIN 3 PD E 5 (AVG'MIN)
AVGMAX 3 PD E 5 (AVG'MAX)

LOCTN1 10 77 ('LOCATION) UNHAND1 4 87 NU Z (ON'HAND)
AVUNT1 4 91 NU Z (AVAIL.'UNITS)
LOCTN2 10 97 UNHAND2 4 107 NU Z AVUNT2 4 111 NU Z
LOCTN3 10 117 UNHAND3 4 127 NU Z AVUNT3 4 131 NU Z
LOCTN4 10 137 UNHAND4 4 147 NU Z AVUNT4 4 151 NU Z

IF NUMBER NE '0021' REJECT ENDIF

LIST NUMBER ITEMNO COMCODE DESCN NOLOCTN UNITMEAS WEIGHT COST
UNITHAND ONORDER AVGUSE AVGMIN AVGMAX WITH 2 AFTER
LIST ' ' AT AVGMAX+6 LOCTN1 UNHAND1 AVUNT1

IF NOLOCTN GE '02'
LIST LOCTN2 AT LOCTN1 UNHAND2 AT UNHAND1 AVUNT2 AT AVUNT1+1
ELSE REJECT ENDIF
IF NOLOCTN GE '03'
LIST LOCTN3 AT LOCTN1 UNHAND3 AT UNHAND1 AVUNT3 AT AVUNT1+1
ELSE REJECT ENDIF
IF NOLOCTN GE '04'
LIST LOCTN4 AT LOCTN1 UNHAND4 AT UNHAND1 AVUNT4 AT AVUNT1+1
ELSE REJECT ENDIF

T1 'COMPUTER ASSOCIATES' T1+102 DYLDATAPAG
T2 'INVENTORY BALANCE REPORT BY VENDOR' WITH 2 AFTER

```

Figure 365 [Example 5](#) Report with Five Detail Lines

COMPUTER ASSOCIATES											DATE	01/15/01	PAGE	1	
INVENTORY BALANCE REPORT BY VENDOR															
VEN NO.	ITEM CODE	COMM CODE	DESCRIPTION	NO OF LOC.	U/M	WGT	COST	TOTAL ON HAND	ON ORDER	AVG USAGE	AVG MIN	AVG MAX	LOCATION	ON HAND	AVAIL. UNITS
0021	09095	23	FEATHER TICKLERS - QUAIL VARIETY	02	EA	001	599.00	30	5	20	5	10			
													TALLALAH	20	2
													WEEHAWKEN	10	
0021	08001	50	BEEES KNEES - BUMBLE	01	FR	002	675.00	15	1	1	6	4			
													MOONACHIE	15	1
0021	09000	10	CRICKET SWITCHES	04	EA	010	995.00	100	1	1	5	20			
													TALLALAH	30	10
													WEEHAWKEN	50	1
													MOONACHIE	15	10
													OOLAGAH	5	5
0021	09100	89	CATS MEOWS - SIAMESE	01	EA	001	999.00	1	20	10	5	100			
													WEEHAWKEN	1	
0021	09015	45	KEYHOLE PEEPERS	02	EA	005	95.00	50	1	5	10	10			
													MOONACHIE	15	10
													OOLAGAH	35	2
0021	09900	15	SOUR GRAPES - BITTER VARIETY	03	BU	015	649.00	25	5	10	3	5			
													TALLALAH	10	
													WEEHAWKEN	6	
													OOLAGAH	9	
0021	08150	08	FAN TANS - PEACOCK VARIETY	01	EA	010	99.00	115	100	20	15	60			
													OOLAGAH	115	100
0021	09120	36	SMOOTH PRUNES - YOUNG RAISINS	03	EA	020	495.00	20	6	25	6	20			
													WEEHAWKEN	10	8
													MOONACHIE	7	5
													OOLAGAH	3	3
0021	09450	30	POLLEN PUFFERS - SNEEZER VARIETY	02	EA	001	445.00	150	15	30	15	50			
													MOONACHIE	55	40
													OOLAGAH	95	60
0021	09060	25	SNOW WHITENER	04	EA	006	590.00	10	6	15	5	8			
													TALLALAH	1	1
													WEEHAWKEN	2	1
													MOONACHIE	2	
													OOLAGAH	5	1

Figure 366 [Example 5](#) Report with Five Detail Lines Results**Example 6 Report with One Detail Line, Two Control Lines, and One Final Line**

Produce an employee-by-cost center report. Detail print each employee, with a final total and control totals by office number and cost center. Use TALLY at all levels.

```

FILE EMPSTCT FB 80 800
COSTCTR 4 1 (COST'CENTER) OFFNO 4 5 (OFFICE'NUMBER)
EMPNO 6 9 (EMPLOYEE'NUMBER) EMPNAME 19 15 (EMPLOYEE'NAME)
HIREDATE 6 34 NU D (HIRE'DATE) SALCLASS 1 40 (SALARY'CLASS)
JOBCLASS 4 41 (JOB'CLASS) LABORGR 1 46 (LABOR'GRADE)
SEX 1 46 MARSTA 2 47 (MRT'ST)
CURRATE 5 49 NU 2 E 4.2 (CURRENT'RATE)
INCDATE 6 54 NU D (INCREASE'DATE) TYPINC 1 60 (TYPE'INC)
INCRATE 5 61 NU 2 E 3.2 (INCREASE'RATE)
WORKAREA
AVG1 5 PD 2 E 3.2 AVG2 5 PD 2 E 3.2
AVG3 5 PD 2 E 3.2 TALLY Z 4

CONTROL OFFNO COSTCTR
LIST COSTCTR OFFNO EMPNO EMPNAME HIREDATE SALCLASS

```

Figure 367 [Example 6](#) Report with One Detail Line, Two Control Lines, and One Final Line (Page 1 of 2)


```

JOBCLASS LABORGR SEX MARSTA CURRATE INCDATE TYPINC INCRATE

ON CHANGE IN OFFNO
AVG1= SUM CURRATE / TALLY
LIST SUM CURRATE SUM INCRATE TALLY (COUNT)
AVG1 (AVERAGE RATE)      '* OFFICE TOTAL' AT EMPNAME WITH 2 AFTER
ON CHANGE IN COSTCTR
AVG2= SUM CURRATE / TALLY
LIST SUM CURRATE SUM INCRATE TALLY (COUNT) AVG2 AT AVG1
    '** COST CENTER TOTAL' AT EMPNAME WITH 2 AFTER
ON FINAL
AVG3= SUM CURRATE / TALLY
LIST SUM CURRATE SUM INCRATE TALLY (COUNT) AVG3 AT AVG1
    '*** GRAND TOTAL' AT EMPNAME WITH EJECT AFTER

T1 'COMPUTER ASSOCIATES, INC.'
T1+1 'VISION:RESULTS'
T1+107 'REPORT NUMBER      DYL511-14'
T2 T2+107 'PERIOD ENDING'
T2+125 DYLDATE
T3 'EMPLOYEE BY COST CENTER' WITH 2 AFTER
T3+107 'PAGE NUMBER'
T3+126 DYLPAGE7

```

Figure 367 [Example 6](#) Report with One Detail Line, Two Control Lines, and One Final Line (Page 2 of 2)

VISION:RESULTS		COMPUTER ASSOCIATES										REPORT			
NUMBER DYL511-14												PERIOD			
ENDING 01/15/01		EMPLOYEE BY COST CENTER										PAGE			
NUMBER 1															
COST CENTER	OFFICE NUMBER	EMPLOYEE NUMBER	EMPLOYEE NAME	HIRE DATE	SALARY CLASS	JOB CLASS	LABOR GRADE	SEX	MRT ST	CURRENT RATE	INCREASE DATE	TYPE INC	INCREASE RATE	COUNT	AVERAGE RATE
OE16	0087	003471	WOLF ROADRUNNER	06/01/90	B	0036	M	M	03	341.00				1	341.00
			* OFFICE TOTAL							341.00					
OE16	0090	003640	YOTO R ROTO	04/14/95	A	0041	M	M	01	500.00					
OE16	0090	004561	TREBOR S ROBERT	10/01/94	B	0041	M	M	01	475.00					
OE16	0090	004777	SUDSY B SUGGINS	02/22/90	C	0036	F	F	05	210.00				3	395.00
			* OFFICE TOTAL							1,185.00					
OE16	0155	005601	THOR VIKING	12/01/93	S	0036	M	M	05	310.00	06/01/95	1	100.00		
OE16	0155	005678	STANLEY YELNATS	06/07/87	S	0036	M	M	02	325.00				2	317.50
			* OFFICE TOTAL							635.00			100.00		
OE16	0170	005831	GODFREY Q ZANE	07/07/92	D	0039	M	M	02	600.00					
OE16	0170	005889	ZAC Z ZOOK	11/25/90	U	0039	F	F	03	340.00				2	470.00
			* OFFICE TOTAL							940.00					
OE16	0190	004361	LORAC B CARROLL	10/10/91	A	0041	F	F	01	365.00	12/01/94	2	20.00		
OE16	0190	006451	DUDLEY G DORIGHT	12/01/84	A	0041	M	M	01	475.00					
OE16	0190	004411	FAMME FAIRWEATHER	08/10/90	S	0041	F	F	01	500.00					
OE16	0190	005512	GINGER A FIZZ	05/17/91	S	0041	F	F	05	295.00				4	408.75
			* OFFICE TOTAL							1,635.00			20.00		
			** COST CENTER TOTAL							4,736.00			120.00	12	394.66
OE17	0155	006636	GUSSIE GASPARD	04/27/88	R	0036	F	F	04	300.00				1	300.00
			* OFFICE TOTAL							300.00					
OE17	0087	005961	SPEEDBALL GONZALES	06/14/90	A	0039	M	M	03	600.00	08/30/94	1	50.00		
OE17	0087	005212	SERMORE HORIZONS	09/01/87	D	0036	M	M	03	265.00	10/25/94	1	75.00		
OE17	0087	004180	BERNIE MILKTOAST	06/06/90	U	0036	M	M	02	150.00				3	338.33
			* OFFICE TOTAL							1,015.00			125.00		
			** COST CENTER TOTAL							1,315.00			125.00	4	328.75
			*** GRAND TOTAL							6,051.00			245.00	16	378.18

Figure 368 [Example 6](#) Report with One Detail Line, Two Control Lines, and One Final Line Results

Example 7 Report with Two Control Lines and One Final Line

Analyze and compute insurance sales commission by area for employees within regional branches. Calculate commissions for areas 1 to 5 based on the following percentages: Area 1=6%, Area 2=5%, Area 3=3%, Area 4=8%, Area 5=10%, and print this along with the employee's base salary, regional branch, and employee number.

```
FILE SALESCOM FB 80 800
SALESMAN 5 1 (SALES'PERSON) SALA1 6 6
NU 2 E 4.2 (AREA'SALES)
SALA2 6 12 NU 2 E 4.2 (AREA'SALES)
SALA3 6 18 NU 2 E 4.2 (AREA'SALES)
SALA4 6 24 NU 2 E 4.2 (AREA'SALES)
SALA5 6 30 NU 2 E 4.2 (AREA'SALES)
BASESAL 6 36 NU 2 E 4.2 (BASE'SALARY) REGBRCH 2 79 (REG'BR)
WORKAREA
AR1 3 PD 2 E 3.2 (1'COMM) AR2 3 PD 2 E 3.2 (2'COMM)
AR3 3 PD 2 E 3.2 (3'COMM) AR4 3 PD 2 E 3.2 (4'COMM)
AR5 3 PD 2 E 3.2 (5'COMM) TCM 4 PD 2 E 4.2 (TOTAL'COMM)
TSL 4 PD 2 E 4.2 (TOTAL'SALES) TOT 4 PD 2 E 4.2 ('TOTAL)
CONTROL SALESMAN REGBRCH
AR1 = SALA1 * 0.06 AR2 = SALA2 * 0.05 AR3 = SALA3 * 0.03
AR4 = SALA4 * 0.08 AR5 = SALA5 * 0.1
TCM = AR1 + AR2 + AR3 + AR4 + AR5
TSL = SALA1 + SALA2 + SALA3 + SALA4 + SALA5
TOT = TCM + BASESAL
ON CHANGE IN SALESMAN
LIST SUPPRESS REGBRCH SUPPRESS SALESMAN SUM SALA1 SUM AR1
SUM SALA2 SUM AR2 SUM SALA3 SUM AR3 SUM SALA4 SUM AR4
SUM SALA5 SUM AR5 SUM TSL SUM TCM SUM BASESAL SUM TOT
WITH 2 AFTER
ON CHANGE IN REGBRCH
LIST SUM SALA1 SUM AR1 SUM SALA2 SUM AR2 SUM SALA3 SUM AR3
SUM SALA4 SUM AR4 SUM SALA5 SUM AR5 SUM TSL SUM TCM
SUM BASESAL SUM TOT WITH 2 AFTER
ON FINAL
LIST SUM SALA1 SUM AR1 SUM SALA2 SUM AR2 SUM SALA3 SUM AR3
SUM SALA4 SUM AR4 SUM SALA5 SUM AR5 SUM TSL SUM TCM
SUM BASESAL SUM TOT WITH 3 AFTER
T1 "SALESPERSON'S COMMISSION JOURNAL" WITH 2 AFTER
T1+1 'DATE' T1+7 DYLDATE
```

Figure 369 [Example 7](#) Report with Two Control Lines and One Final Line

DATE 01/15/01		SALESPERSON'S COMMISSION JOURNAL													
REG	SALES	AREA	1	AREA	2	AREA	3	AREA	4	AREA	5	TOTAL	TOTAL	BASE	TOTAL
BR	PERSON	SALES	COMM	SALES	COMM	SALES	COMM	SALES	COMM	SALES	COMM	SALES	COMM	SALARY	
01	21683									4,622.46	462.24	4,622.46	462.24		462.24
01	32861			900.47	45.02	89.81	2.68					989.98	47.70	52.00	99.70
01	41897							761.89	60.95			761.89	60.95		60.95
01	42121	73.69	4.42	82.20	4.11							155.89	8.53	30.00	38.53
01	46044			100.00	5.00	300.00	9.00	537.68	43.01	200.00	20.00	1,137.68	77.01	240.00	317.01
01	58210	1,051.88	63.10			318.00	9.54					1,369.88	72.64	200.00	272.64
		1,125.57	67.52	1,082.67	54.13	707.51	21.22	1,299.57	108.96	4,822.46	482.24	9,037.78	729.07	522.00	1,251.07
02	28132					210.12	6.30					210.12	6.30	111.50	117.80

Figure 370 [Example 7](#) Report with Two Control Lines and One Final Line
Results (Page 1 of 2)

02	29516									2,897.15	289.71	2,897.15	289.71	70.00	359.71
02	30001	2.00	.12	418.21	20.91	598.32	17.94					1,018.53	38.97	20.00	58.97
02	38976							389.70	31.17	931.62	93.16	1,321.32	124.33		124.33
		2.00	.12	418.21	20.91	808.44	24.24	389.70	31.17	3,828.77	382.87	5,447.12	459.31	201.50	660.81
								1,127.57	67.64	1,500.88	75.04	1,515.95	45.46	1,689.27	138.13
								8,651.23	865.11	4,484.90	1,188.38	723.50	1,911.88		

Figure 370 [Example 7](#) Report with Two Control Lines and One Final Line Results (Page 2 of 2)

Example 8 Report with One Detail Line, One Control Line, and One Final Line with Two Output Files

Generate an open item accounts receivable report, along with two output files for use in later generating confirmation letters and labels for a national sales company.

```

FILE OPITARC FB 125 2500
IN97BYTE 97 1 CUSTNO 5 1 (CUST.'NUMBER)
CUSTNAME 24 6 (CUST.'NAME) TRANDATE 6 102 NU D (TR.'DATE)
INVNO 5 108 (INVOICE'NUMBER) DESCODE 4 113 (DESC.'CODE)
TOTAMT 6 117 PD 2 E 5.2 (TOTAL'AMOUNT)
CRDTLMT 4 98 NU (CREDIT'LIMIT)
FILE JFILE OUTPUT FROM KFILE FB 97 970
FILE KFILE OUTPUT FROM KFILE FB 105 1050
KFL97 97 1 OUTTOTAM 8 98 PD 5
WORKAREA
DY1T30 6 PD 2 E 5.2 (1-30'DAYS)
DY30T60 6 PD 2 E 5.2 (30-60'DAYS)
DY60T90 6 PD 2 E 5.2 (60-90'DAYS)
DY0V90 6 PD 2 E 5.2 (90-OVR'DAYS)
REPTDATE 8 VALUE '07/02/89' THIRTYDY 6 VALUE '890602'
SIXTYDAYS 6 VALUE '890502' NINETYDY 6 VALUE '890402'
CONTROL CUSTNO
ON ONE MOVE IN97BYTE TO KFL97 ENDONE
**** REINITIALIZE BUCKETS *****
DY1T30 = 0 DY30T60 = 0 DY60T90 = 0 DY0V90 = 0
IF TRANDATE GT THIRTYDY MOVE TOTAMT TO DY1T30 GOTO PR ENDIF
IF TRANDATE GT SIXTYDAYS MOVE TOTAMT TO DY30T60 GOTO PR ENDIF
IF TRANDATE GT NINETYDY MOVE TOTAMT TO DY60T90 GOTO PR ENDIF
MOVE TOTAMT TO DY0V90
PR: LIST SUPPRESS CUSTNO SUPPRESS CUSTNAME TRANDATE
INVNO DESCODE DY1T30 DY30T60 DY60T90 DY0V90 TOTAMT
SUPPRESS CRDTLMT
ON CHANGE IN CUSTNO
IF SUM TOTAMT GT 0
WRITE JFILE
MOVE SUM TOTAMT TO OUTTOTAM
WRITE KFILE ENDIF
MOVE IN97BYTE TO KFL97
LIST SUM DY1T30 SUM DY30T60 SUM DY60T90
SUM DY0V90 SUM TOTAMT
'*CUST.TOTAL' AT CUSTNAME+1 WITH 2 AFTER
ON FINAL
LIST SUM DY1T30 SUM DY30T60 SUM DY60T90
SUM DY0V90 SUM TOTAMT
'***GRAND TOTAL' AT CUSTNAME+1
T1 'NOTROM SALES COMPANY'
T2 'OPEN ITEM AGED RECEIVABLE REPORT'
T3 'WITH 2 AFTER T3+63 REPTDATE
T4 FOOTING T4+61 DYLPAGE

```

Figure 371 [Example 8](#) Report with One Detail Line, One Control Line, and One Final Line with Two Output Files

NOTROM SALES COMPANY OPEN ITEM AGED RECEIVABLE REPORT 01/15/01										
CUST. NUMBER	CUST. NAME	TR DATE	INVOICE NUMBER	DESC CODE	1-30 DAYS	30-60 DAYS	60-90 DAYS	90-OVR DAYS	TOTAL AMOUNT	CREDIT LIMIT
00010	DAGWOOD STUMSTEAD	95/10/15	71380	ADJS			25.95-		25.95-	0500
	*CUST. TOTAL	95/10/15	71380	INV			25.95		25.95	
00020	ELMER BUDD ESQUIRE	95/11/12	71434	INV		23.66			23.66	0100
	*CUST. TOTAL					23.66			23.66	
00125	DR LUCY LOUDMOUTH	95/04/30	71400	PMT			1.05		1.05	0100
		95/05/10	71428	ADJA		2.00			2.00	
		95/04/30	71405	ADJS			1.05-		1.05-	
	*CUST. TOTAL					2.00			2.00	
00150	FRED BARON	95/04/20	71382	INV			64.94		64.94	0700
		95/06/10	71382	PMT	64.95				64.95	
	*CUST. TOTAL				64.95		64.94		129.89	
00175	SYLVESTER KAT	95/04/01	71340	INV				987.95	987.95	1000
	*CUST. TOTAL							987.95	987.95	
00190	GORDO LARD	95/11/30	71460	PMT		19.45			19.45	0200
	*CUST. TOTAL					19.45			19.45	
00205	GROOM HILDA	95/10/20	71387	PMT			109.60		109.60	0500
		95/11/10	71425	ADJS		99.84-			99.84-	
		95/11/10	71475	INV		799.55			799.55	
	*CUST. TOTAL					699.71	109.60		809.31	
00600	LUNIS BLANKET	95/10/29	71399	PMT			10.03		10.03	0100
		95/11/01	71418	ADJA			10.03-		10.03-	
	*CUST. TOTAL									
00820	RUGS RABBIT	95/11/01	71415	INV			.50		.50	0050
	RUGS RABBIT	95/11/10	71430	PMT		5.00			5.00	
	*CUST. TOTAL					5.00	.50		5.50	
00922	FLUITO CANINE	95/10/20	71345	ADJS			50.95-		50.95-	0500
		95/10/10	71345	PMT			50.95		50.95	
		95/10/30	71408	INV			5.00		5.00	
	*CUST. TOTAL						5.00		5.00	
	**GRAND TOTAL				64.95	749.82	180.04	987.95	1,982.76	
					PAGE 1					

Figure 372 [Example 8](#) Report with One Detail Line, One Control Line, and One Final Line with Two Output Files Results

Example 9 Report in Fixed Print Position

Generate a confirmation letter of the amount due and receivable for each California customer. There are two input files — one containing the entire letter and the other containing the file produced from [Example 8](#) that contains the pertinent information.

```

FILE SYSIN CARDS STATUS LTREOF
INLTR 70 1
FILE OPITARCV INPUT FB
CUSTNO 5 1 CUSTNAME 24 6 COMPANY 24 30 ADDRESS 24 54
CITY 13 78 STATE 2 91 ZIP 5 93 AMT 8 98 PD 5 E 4.2
WORKAREA
LTRAREA 840 LTRLINE1 70 1 LTRLINE2 70 LTRLINE3 70
LTRLINE4 70 LTRLINE5 70 LTRLINE6 70 LTRLINE7 70
LTRLINE8 70 LTRLINE9 70 LTRLINE10 70 LTRLINE11 70

REPORT 40 LONG

```

Figure 373 [Example 9](#) Report in Fixed Print Position (Page 1 of 2)

```

RDFL1: READ SYSIN
      IF LTREOF EQ 'E' GOTO RDFL2 ENDIF
      MOVE INLTR TO LTRLINE1 (INW)
      INW = INW + 70
      IF INW LT 840 GOTO RDFL1 ENDIF
      MOVE 'TBL EXCEED' TO DYLPRTCOMM
      PRINT SYSIN STOP

RDFL2: READ OPITARC
      IF STATE NE 'CA' REJECT ENDIF
      LIST CUSTNO AT 15
      LIST CUSTNAME AT 15
      LIST COMPANY AT 15
      LIST ADDRESS AT 15
      LIST CITY AT 15 STATE AT 29 ZIP AT 32 WITH 03 AFTER
      LIST LTRLINE1 AT 15 WITH 02 AFTER
      LIST LTRLINE2 AT 15 WITH 02 AFTER
      LIST LTRLINE3 AT 15
      LIST LTRLINE4 AT 15
      LIST LTRLINE5 AT 15 WITH 02 AFTER
      LIST LTRLINE6 AT 15 AMT AT 67
      LIST LTRLINE7 AT 15
      LIST LTRLINE8 AT 15
      LIST LTRLINE9 AT 15 WITH 02 AFTER
      LIST LTRLINE10 AT 15 WITH 03 AFTER
      LIST LTRLINE11 AT 15 WITH EJECT AFTER

T1
T1+55 'Notrom Sales Company'
T2
T2+55 'Liverpool Lane   #3126'
T3 WITH 3 AFTER
T3+55 'Green Pastures, Conn'
T4 FOOTING  T4+36 'Confirmation'
T5 FOOTING  WITH 2 AFTER
T5+15 'The above information is corre' T5+45 'ct except as noted'
T6 FOOTING  T6+15 'Signed' T6+42 'Date'
FIN
Dear Customer:
Auditors Confirmation
  This form is being sent to you to enable our independent
auditors to confirm the correctness of our records. It is not
a request for payment. However if you wish to remit please do.
  Our records on January 15, 1998 show an amount of $
receivable from you. Please confirm that this agrees with your
records on that date by signing it and returning it directly
to our auditors.

      Thank you very much
      NOTSVILLE NOTROM

```

Figure 373 [Example 9](#) Report in Fixed Print Position (Page 2 of 2)

Notrom Sales Company
Liverpool Lane #3126
Green Pastures, Conn

00125
DR LUCY LOUDMOUTH
HELPME INC.
5 CENTS AVE
HELPPVILLE CA 94024

Dear Customer:

Auditors Confirmation

This form is being sent to you to enable our independent auditors to confirm the correctness of our records. It is not a request for payment. However if you wish to remit please do.

Our records on January 15, 1998 show an amount of \$ 2.00 receivable from you. Please confirm that this agrees with your records on that date by signing it and returning it directly to our auditors.

Thank you very much,

NOTSVILLE NOTROM

Confirmation
The above information is correct except as noted

Signed _____ Date _____

Figure 374 [Example 9](#) Report in Fixed Print Position Results

Example 10 Four-up Report in Fixed Print Position

Produce four-up name and address labels for all California accounts due and receivable. Use the output file created in [Example 8](#) as the input.

```
FILE OPITARCV FB 97 9700
CUSTOMER 97 1 CITY 13 78 STATE 2 91 ZIP 5 93
WORKAREA
LTRAREA 400 LBLCTR 2 401 PD RCRD1 97 1 RCRD2 97 101
RCRD3 97 201 RCRD4 97 301 NMEA 24 6 CMPYA 24 30
ADDA1 24 54 ADDA2 15 78 ZIPA 5 93 NMEB 24 106
CMPYB 24 130 ADDB1 24 154 ADDB2 15 178 ZIPB 5 193
NMEC 24 206 CMPYC 24 230 ADDC1 24 254 ADDC2 15 278
ZIPC 5 293 NMED 24 306 CMPYD 24 330 ADDD1 24 354
ADDD2 15 378 ZIPD 5 393

IF STATE NE 'CA' REJECT ENDIF
IF LBLCTR EQ 0
MOVE CUSTOMER TO RCRD1 GOTO ADD ENDIF
IF LBLCTR EQ 1
MOVE CUSTOMER TO RCRD2 GOTO ADD ENDIF
IF LBLCTR EQ 2
MOVE CUSTOMER TO RCRD3 ELSE GOTO CK4 ENDIF
ADD: LBLCTR = LBLCTR + 1 REJECT

CK4: MOVE CUSTOMER TO RCRD4
LBLCTR = 0
LSTST:
```

Figure 375 [Example 10](#) Four-up Report in Fixed Print Position (Page 1 of 2)

```

LIST NMEA AT 1 NMEB AT 35 NMEC AT 69 NMED AT 103
LIST CMPYA AT 1 CMPYB AT 35 CMPYC AT 69 CMPYD AT 103
LIST ADDA1 AT 1 ADDB1 AT 35 ADDC1 AT 69 ADDD1 AT 103
LIST ADDA2 AT 1 ZIPA AT 18 ADDB2 AT 35 ZIPB AT 52
      ADDC2 AT 69 ZIPC AT 86 ADDD2 AT 103 ZIPD AT 120
      WITH 03 AFTER
MOVE SPACES TO LTRAREA
LSTEN:
ON END OF INPUT PERFORM LSTST TO LSTEN STOP

```

Figure 375 [Example 10](#) Four-up Report in Fixed Print Position (Page 2 of 2)

```

DR. LUCY LOUMOUTH      FRED BARON      SYLVESTER KAT      WALLY COYOTE
HELME INC              FLIGHT GEAR INC BIRDSEED CO        BEEP BEEP INC
5 CENTIS AVE           123 DOGHOUSE LANE 55 TWEETIE LANE    999 ROADRUNNER IN
HELPVILLE CA 94024   SNOOPVILLE CA 90100 CATSVILLE CA 96050  SAGEBRUSH CA 95120

BUGS RABBIT           PLUTO CANINE
VEGMATIC INC          MICKEY RODENT INC
1642 CARROT LANE      177 FLOPPYEAR AVE
BUCKSVILLE CA 95432  BOWWOW CA 90024

```

Figure 376 [Example 10](#) Four-up Report in Fixed Print Position Results**Example 11 Report with One Detail Line and One Control Line with Fixed Print Positions**

Produce a report using both a fixed print position for the data and a fixed print position for the column headings.

```

FILE ACCTG COUNT RECCTR
TRANS 7 4 NAME 25 85 ACCOUNT 2 182 BALANCE 5 170 PD 2 A
CONTROL ACCOUNT
LIST SUPPRESS ACCOUNT AT 3 TRANS AT 14
NAME AT 26 BALANCE AT 57
REPORT
ON CHANGE IN ACCOUNT
LIST ACCOUNT AT 3 SUM BALANCE AT 55 WITH 1 BEFORE AND 2 AFTER
T1 'REPORT' FIXED WITH 2 AFTER
T2
'ACCOUNT      TRANS      NAME      BALANCE '
FIXED WITH 2 AFTER

```

Figure 377 [Example 11](#) Report with One Detail Line and One Control Line with Fixed Print Positions for Data

The following text is only a partial illustration of the first and last sections of the report.

```

REPORT
ACCOUNT      TRANS      NAME      BALANCE
BO           8006547    TORRES, ERNESTO      44.99
BO
EO           6208657    CHO PYUNG, SUH        32.00
           7082509    S.F.MEM.HOSP.         5.00
           6002587    SANTA FE HOSP ASSN    15.00
EO
FO           6107265    GENVARDI, G J         43.00
           6095631    TODIPE, MICHAEL       45.24

```

Figure 378 [Example 11](#) Report with One Detail Line and One Control Line with Fixed Print Positions Results (Page 1 of 2)

	6123228	SILVA, JULIAN	.00
	6059708	CHAVEZ, RAY	15.00
	6044395	CANO, MICHAEL S	15.00
	6024963	HILL, GARY E	3.80
	8011508	HUGHES, RAY	178.70
FO			300.74
IO	2002299	PLACIDO, ORTEGA	413.58
	6123317	VASGUEZ, IRENE	.00
	6112536	CHAVEZ, NORMA A	55.00
	6009166	LOCKE, JEFFREY	15.00
	8012644	MONTEZ, CARMEN	89.28
	6218113	AGUIERA, EMILIO	108.44
IO			681.30
KO			3,765.92
LO	6114113	NERY, GENEROSO	45.24
LO			45.24
MA	7099657	MALTSBERGER, JOHN A	6.72
	7102194	TURNER, HAROLD	5.12
	7106246	SMITH, AUSTIN	5.12
	9017828	HILL, MYRTLE G	11.20
	9000534	HOPKINS, BARRY P	14.00
	9002626	WORRELL, TED	13.00
	6215785	RODEN, HAROLD	170.91
	6208924	WHITE, ELMER	15.00
	6211151	COX, WILLIAM	100.20
MA			341.27

Figure 378 [Example 11](#) Report with One Detail Line and One Control Line with Fixed Print Positions Results (Page 2 of 2)

Example 12 Characters per Line Override on REPORT

Produce a report from instream data. The report should be only 58 characters wide. A control break is specified for a change in department.

```
FILE SYSIN INPUT CARDS
DEPARTMENT 3 NU SSN 9 NU S NET 7 NU 2 Z (NET'PAY) NAME 10
REPORT 58 WIDE 55 LONG
CONTROL DEPARTMENT
LIST NAME SSN NET
ON CHANGE IN DEPARTMENT
LIST 'DEPARTMENT TOTAL' AT NAME SUM NET
WITH 1 BEFORE AND 2 AFTER
T1 'EMPLOYEES BY DEPARTMENT' WITH 2 AFTER
FIN
(instream data)
```

Figure 379 [Example 12](#) Characters per Line Override on REPORT

EMPLOYEES BY DEPARTMENT		
NAME	SSN	NET PAY
GREGORY	879-45-6161	651.23
ROGERS	546-24-5161	856.94
HOLDEN	452-89-9818	245.62
PARTRIDGE	623-54-6121	1452.00
CARPENTER	989-55-2123	900.21

Figure 380 [Example 12](#) Characters per Line Override on REPORT Results

NORRIS	321-45-6213	524.56
CROSLAND	235-65-1552	1000.00
BOND	089-01-0070	654.21
SELLERS	546-25-6135	1520.00
DOUGLAS	632-15-1595	546.82
DEPARTMENT TOTAL		8351.59
REDLANDS	452-81-9821	499.65
MCDONALD	562-26-1515	610.15
FOSTER	632-12-1584	365.25
DEPARTMENT TOTAL		1475.05
RIGBY	589-93-6342	405.01
ACKERMAN	782-92-5660	365.25
GOMEZ	525-65-3245	800.41
DEPARTMENT TOTAL		1570.67
DESOTO	213-65-8888	502.52
LEE	222-45-6612	425.65
DEPARTMENT TOTAL		928.17

Figure 380 [Example 12](#) Characters per Line Override on REPORT Results**Example 13 Report with One Detail Line with LIST Subordinate to IF**

Generate a detail report for department 911. Use the LIST statement subordinate to the IF. This is valid only at detail time.

```
FILE SYSIN INPUT CARDS
DEPARTMENT 3 NU SSN 9 NU S NET 7 NU 2 Z (NET'PAY) NAME 10
REPORT 58 WIDE 55 LONG
IF DEPARTMENT EQ 911
  LIST NAME SSN NET ENDIF
T1 'EMPLOYEES IN DEPARTMENT 911' WITH 2 AFTER
FIN
(instream data)
```

Figure 381 [Example 13](#) Report with One Detail Line with LIST Subordinate to IF

EMPLOYEES IN DEPARTMENT 911		
NAME	SSN	NET PAY
GREGORY	879-45-6161	651.23
ROGERS	546-24-5161	856.94
HOLDEN	452-89-9818	245.62
PARTRIDGE	623-54-6121	1452.00
CARPENTER	989-55-2123	900.21
NORRIS	321-45-6213	524.56
KINSLEY	235-65-1552	1000.00
BOND	089-01-0070	654.21
SELLERS	546-25-6135	1520.00
DOUGLAS	632-15-1595	546.82

Figure 382 [Example 13](#) Report with One Detail Line with LIST Subordinate to IF Results

Example 14 Report with Title and Footing Modification

Generate a report of all departments. Page eject whenever a department number changes. Use title and footing modifications to identify the department on each page.

```
FILE SYSIN INPUT CARDS
DEPARTMENT 3 NU SSN 9 NU S NET 7 NU 2 Z (NET'PAY) NAME 10
WORKAREA
  DEPTFOOT 3 DEPTCHANGE 1 VALUE 'Y'
CONTROL DEPARTMENT
IF DEPTCHANGE EQ 'Y'
  MOVE DEPARTMENT TO DEPTFOOT
  MOVE 'N' TO DEPTCHANGE ENDIF
LIST NAME SSN NET
ON CHANGE IN DEPARTMENT
  MOVE 'Y' TO DEPTCHANGE
  LIST 'DEPARTMENT TOTAL' AT NAME SUM NET
  WITH 1 BEFORE AND EJECT AFTER
T1 WITH 2 AFTER
T1+50 'EMPLOYEES BY DEPARTMENT'
T1+74 DEPARTMENT
T2 FOOTING T3 FOOTING
T3+50 'EMPLOYEES IN DEPARTMENT'
T3+74 DEPTFOOT
FIN
(instream data)
```

Figure 383 [Example 14](#) Report with Title and Footing Modification

EMPLOYEES BY DEPARTMENT 911		
NAME	SSN	NET PAY
GREGORY	879-45-6161	651.23
ROGERS	546-24-5161	856.94
HOLDEN	452-89-9818	245.62
PARTRIDGE	623-54-6121	1452.00
CARPENTER	989-55-2123	900.21
NORRIS	321-45-6213	524.56
CROSLAND	235-65-1552	1000.00
BOND	089-01-0070	654.21
SELLERS	546-25-6135	1520.00
DOUGLAS	632-15-1595	546.82
DEPARTMENT TOTAL		8351.59
EMPLOYEES IN DEPARTMENT 911 EMPLOYEES BY DEPARTMENT 925		
NAME	SSN	NET PAY
REDLANDS	452-81-9821	499.65
MCDONALD	562-26-1515	610.15
FOSTER	632-12-1584	365.25
DEPARTMENT TOTAL		1475.05
EMPLOYEES IN DEPARTMENT 925		

Figure 384 [Example 14](#) Report with Title and Footing Modification Results
(Page 1 of 2)

```

EMPLOYEES BY DEPARTMENT 931

NAME                      SSN                      NET
                        PAY
RIGBY                     589-93-6342         405.01
ACKERMAN                  782-92-5660         365.25
GOMEZ                     525-65-3245         800.41
DEPARTMENT TOTAL                                1570.67

EMPLOYEES IN DEPARTMENT 931

EMPLOYEES BY DEPARTMENT 965

NAME                      SSN                      NET
                        PAY
DESOTO                    213-65-8888         502.52
LEE                       222-45-6612         425.65
DEPARTMENT TOTAL                                928.17

EMPLOYEES IN DEPARTMENT 965

```

Figure 384 [Example 14](#) Report with Title and Footing Modification Results
(Page 2 of 2)

Report Print Line Exit—Your Exit Routines

When using VISION:Results, you might want to examine a report line before it is written and either eliminate it or write it to another device, in addition to or instead of writing it to the report. VISION:Results provides an exit point, using the REPORT statement, from the report writer to your specified external subroutine. This exit occurs immediately before the print line is written. The report lines can be detail lines, control break lines, titles, or blank lines.

Procedure

To start the exit, specify the external subroutine's module name in the REPORT statement.

```
REPORT modulename
```

Figure 385 Exit Routine

The module name is the phase name in VSE and member name in z/OS of your written subroutine that interrogates the print line. The subroutine must have been previously link edited to a Load or Core Image Library. The subroutine module is loaded into memory prior to execution and is entered before each print line is written.

z/OS: The member name must be DYLPRCx, where x is an alphabetic character (B through Z).

REPORT DYLPRCB

VSE: The phase name must be DYL280Rx, where x is an alphabetic character (B through F). The character indicates the external subroutine's requirement.

B = User-written, self-relocating, exit module has a maximum size of 2K.

C = User-written, self-relocating, exit module has a maximum size of 4K.

D = User-written, self-relocating, exit module has a maximum size of 8K.

E = User-written, self-relocating, exit module has a maximum size of 63K.

F = User-written, self-relocating, exit module of any size. VISION:Results CDLOADs; use GETVIS to acquire storage.

REPORT DYL280RB
REPORT 100 WIDE DYL280RC

Figure 386 Exit Routine Example

After examination of the print line, if you do not want to print the line, a value of X'FF' must be returned to VISION:Results in byte 1 of the print line. This value is set by your program. A value of X'FE' is placed in byte 1 of the print line area after all report lines have been passed to your routine. This provides you with the opportunity to do final processing before returning to VISION:Results.

VISION Results allows report print line exit routine DYL280RF to be of any size, which the system loads using CDLOAD instead of LOAD. Use GETVIS to acquire storage instead of using the space after the end of your program, making sure the program is link edited to run as AMODE(24).

Considerations for Writing Your Report Line Exit

- A parameter list is passed to your program. Register 1 points to this list. The parameter list contains the following:
 - address of the current print line
 - address of a VISION:Results data area (PERMDATA) containing addresses of record and work areas, DCB & DTF addresses, and so onMost of your routines probably do not require this data.
- Do not open/close SYS280R (z/OS and VM) or SYSLST (VSE). This is done by VISION:Results.
- If you write your own report file, you are responsible for opening and closing that file.

- The VISION:Results registers must be saved and restored by your exit routine. An 18-fullword save area is provided by VISION:Results and its address is passed in register 13.
- z/OS
The library in which the subroutine resides must be identified by a JOBLIB or STEPLIB statement if not on an automatically searched library.
- z/OS and VM
The default record length (LRECL) for the SYS280R report data set is normally the length of the characters per line plus 1 (carriage control). You should be aware of this when you are passed a report line. For example, if 85 characters-per-line is specified for a report, the record length is 86 bytes long. When a report line is passed to your exit routine, there may be unrecognizable data beyond byte 86. However, you can prevent shortened records by specifying a record length (for example, 133) in the SYS280R DD (SYS280R FILEDEF) statement that overrides the default length.
- VSE
The subroutine must be self-relocating, or the system must have a relocation package or a relocating loader. The phase statement used to link edit normally looks like this:

```
PHASE DYL280RB,+0
```

Figure 387Exit Routine VSE

Chapter 17: XML Support

Extensible Markup Language (XML) provides a way for different platforms to communicate by transferring information from one to another. Using XML you can export VISION:Results XML formatted data from the mainframe to your PC. XML allows you to describe, define, create, and transmit data in any conceivable format. VISION:Results enables you to create a simple XML document that consists of multiple rows of a list of selected data elements.

The benefits of XML are:

- Applications can easily read information from a variety of platforms. This simplifies the sharing of data between users.
- XML data can be read without having detailed knowledge of how that data is structured. The DTD or schema provides this information.
- Data is tagged in a way that allows addition or removal of elements without impacting existing elements, thus making it easier to change the content and structure of data without changing the application.

Using XML

To allow the creation of XML data and schema documents, new commands—XMLFILE and XMLWRITE—have been added to VISION:Results. Multiple XMLFILE and XMLWRITE command sets may be coded in a single program. The XML data and schema documents are created by a C++ environment API, which is referred to as the XML API. The resulting XML data and schema documents are found in output files defined by the programmer. The files can be defined as either sequential data sets or HFS path files. A PDSE file can also be used as a sequential data set when the member name is included in the JCL. The PDSE files and sequential data sets must have RECFM=VB and a minimum LRECL of 255.

Important! VISION:Results does not support the use of PDS files, because they do not allow the update of multiple members at the same time.

XMLFILE Syntax

```
XMLFILE ddname [STATUS dataname-s] FIELDS !dataname-f!
```

XMLFILE ddname

The ddname following XMLFILE identifies the JCL DD statement that contains the XML output data document. An XML output schema document is also created. An output schema JCL DD statement with the same XML output data ddname but with an S at the end needs to be added to the JCL, with the same file requirements used for the output data file.

For the XMLFILE statement to function successfully, the following conditions must be met:

- The output schema ddname must be present. (The DD statement may point to DUMMY if the schema is not required.)
- The output data ddname must be seven characters or less in order to add an S for the output schema ddname. Existing DDNAMES will not be duplicated when these DDNAMES are generated.
- The XMLFILE statement must precede an XMLWRITE for a given ddname.
- The output schema and data ddname files must be allocated when the XMLWRITE is issued.
- The dataname-f following the FIELDS keyword identifies one or more previously defined datanames that constitutes the XML document.
- The maximum number of dataname fields is 899.
- The dataname-s following the STATUS keyword establishes a 1-byte character field that contains the file status. The STATUS keyword is optional and *must not be a previously defined dataname*. It should be used only if the programmer does *not* want VISION:Results to perform the default error handling when an error is returned from the XML API. For more information, see [Error Handling on page 447](#)

XMLWRITE ddname

When the XMLWRITE command is executed, a record is written to the data file specified by the ddname originally described on the XMLFILE statement above. The entire schema file is written when the first write statement is encountered. The XMLFILE statement must precede an XMLWRITE for a given ddname.

Sample Program

The following is a sample program using the XMLFILE and XMLWRITE command set.

```
FILE ARFILE FB 352
COPY ARFIELDS
XMLFILE XMLOUT FIELDS ACCOUNT NAME ADD1 BALANCE
XMLWRITE XMLOUT
```


Creating XML Documents Using HFS Path

The following is an example of how an XML document is created using an HFS path.

```
//XMLFILE EXEC PGM=DYL280,REGION=0M
//STEPLIB DD DISP=SHR,DSN=YOUR.RESULTS.LOAD.LIBRARY
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(4,4))
//SYSPRINT DD SYSOUT=*
//SYS280R DD SYSOUT=*
//ARFILE DD DISP=SHR,DSN=YOUR.RESULTS.DEMOFILE
//SYSCOPY DD DISP=SHR,DSN=YOUR.RESULTS.SOURCE.LIBRARY
//***** XML STYLESHEET *****
//XMLXSL DD DISP=SHR,DSN=YOUR.RESULTS.HTMLLIB(XSL1)
//*
//XMLEUT DD PATH='/your/XML/path/data.xml',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRUSR,SIWUSR)
//XMLEUTS DD PATH='/your/XML/path/schema.xsd',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRUSR,SIWUSR)
//SYSIN DD *
FILE ARFILE FB 352
COPY ARFIELDS
XMLFILE XMLEUT FIELDS ACCOUNT NAME ADD1 BALANCE
XMLWRITE XMLEUT
/*
//
```

Figure 388 Create an XML Document Using HFS Path

The HFS pathname in the JCL example shown in [Figure 388](#) must be enclosed within single quotation marks. You must use the extensions xml and xsd for the XML data and schema documents, respectively. When the XMLXSL DD statement is present, VISION:Results automatically allocates an HFS path name similar to the HFS path name but with an XSL ending qualifier. For example, if the HFS path name for the data document is

mydata.xml

then the HFS path name for the style sheet would be as follows:

/your/XML/path/mydata.xml.xsl

When the XML API allocates the HFS path name for a style sheet, it internally specifies a DDNAME similar to the data document, ending with an L. The style sheet DDNAME must not be coded in the JCL or a style sheet will not be created.

Creating XML Documents Using PDSE

The following is an example of how an XML document is created using PDSE.

```
//XMLFILE EXEC PGM=DYL280,REGION=0M
//STEPLIB DD DISP=SHR,DSN=YOUR.RESULTS.LOAD.LIBRARY
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(4,4))
//SYSPRINT DD SYSOUT=*
//SYS280R DD SYSOUT=*
//ARFILE DD DISP=SHR,DSN=YOUR.RESULTS.DEMOFILE
//SYSCOPY DD DISP=SHR,DSN=YOUR.RESULTS.SOURCE.LIBRARY
//***** XML STYLESHEET *****
//XMLXSL DD DISP=SHR,DSN=YOUR.RESULTS.HTMLLIB(XSL1)
//*
//XMLOUT DD DSN=YOUR.XML.PDSE(data),DISP=SHR
//XMLOUTS DD DSN=YOUR.XML.PDSE(schema),DISP=SHR
//SYSIN DD *
FILE ARFILE FB 352
COPY ARFIELDS
XMLFILE XMLOUT FIELDS ACCOUNT NAME BALANCE
XMLWRITE XMLOUT
/*
//
```

Figure 389 Create an XML Document Using PDSE

When the XMLXSL DD statement is present, VISION:Results dynamically allocates a PDSE member name for the style sheet in the same PDSE as the data document. The style sheet member name will be similar to the member name used for the data document, but ending with an L. For instance, if the PDSE member name for the data document is

MYDATA

then the PDSE member name for the style sheet would be as follows:

YOUR.XML.PDSE(MYDATAL)

Note: If the length of the XML member name is greater than seven characters, then the last character of the XSL name is replaced with an L.

When the XML API allocates the PDSE member name for a style sheet, it internally specifies a DDNAME similar to the data document, ending with an L. The style sheet DDNAME must not be coded in the JCL or a style sheet will not be created.

Creating XML Documents Using Sequential Data Sets

The following is an example of how an XML document is created using sequential data sets.

```
//XMLFILE EXEC PGM=DYL280,REGION=0M
//STEPLIB DD DISP=SHR,DSN=YOUR.RESULTS.LOAD.LIBRARY
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(4,4))
//SYSPRINT DD SYSOUT=*
//SYS280R DD SYSOUT=*
//ARFILE DD DISP=SHR,DSN=YOUR.RESULTS.DEMOFILE
//SYSCOPY DD DISP=SHR,DSN=YOUR.RESULTS.SOURCE.LIBRARY
//XMLEOUT DD DSN=YOUR.XML.DATA,DISP=(,CATLG),
// UNIT=SYSDA,SPACE=(TRK,(15,15)),
// DCB=(RECFM=VB,LRECL=8024,BLKSIZE=8028)
//XMLEOUTS DD DSN=YOUR.XML.SCHEMA,DISP=(,CATLG),
// UNIT=SYSDA,SPACE=(TRK,(15,15)),
// DCB=(RECFM=VB,LRECL=8024,BLKSIZE=8028)
//SYSIN DD *
FILE ARFILE FB 352
COPY ARFIELDS
XMLFILE XMLEOUT FIELDS ACCOUNT NAME ADD1 BALANCE
XMLWRITE XMLEOUT
/*
//
```

Figure 390 Create XML Data and Schema Documents Using Sequential Data Sets

VISION:Results does not create a style sheet when the XML data and schema documents are written to sequential data sets. In this case, the web browser default style sheet will be used to format and view the document.

Dynamic Allocation of XML Documents Using HFS Path

The following is an example of dynamic allocation of XML documents using an HFS path.

```
//XMLFILE EXEC PGM=DYL280,REGION=0M
//STEPLIB DD DISP=SHR,DSN=YOUR.RESULTS.LOAD.LIBRARY
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(4,4))
//SYSPRINT DD SYSOUT=*
//SYS280R DD SYSOUT=*
//ARFILE DD DISP=SHR,DSN=YOUR.RESULTS.DEMOFILE
//SYSCOPY DD DISP=SHR,DSN=YOUR.RESULTS.SOURCE.LIBRARY
//***** XML STYLESHEET *****
//XMLXSL DD DISP=SHR,DSN=YOUR.RESULTS.HTMLLIB(XSL1)
//*
//SYSIN DD *
*****
* DYNAMIC ALLOCATION JCL DD STATEMENTS ARE BELOW.
*****
WORKAREA
  JCLSTAT 1
  MSG1 41 VALUE 'SYNTAX ERROR IN DYNAMICALLY ALLOCATED'
  'JCL'
  MSG2 47 VALUE 'ALLOCATION FAILURE IN DYNAMICALLY'
  'ALLOCATED JCL'
WORKAREA
  OUTJCL 480 1
  FILLER 80 1 VALUE
  " //XMLEOUT DD PATH='/your/xml/path/data.xml', "
```

Figure 391 JCL to Dynamically Allocate Files Using HFS Path (Page 1 of 2)

```
          FILLER              80  VALUE
"// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),"
          FILLER              80  VALUE
"// PATHMODE=(SIRUSR,SIWUSR)"
          FILLER              80  VALUE
"//XMLOUTS DD PATH='/your/xml/path/schema.xsd',"
          FILLER              80  VALUE
"// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),"
          FILLER              80  VALUE
"// PATHMODE=(SIRUSR,SIWUSR)"

ON ONE

ALLOCATE OUTJCL STATUSFLAG JCLSTAT
IF JCLSTAT EQ 'N'
  PRINT MSG1
  STOP
ENDIF
IF JCLSTAT EQ 'F'
  PRINT MSG2
  STOP
ENDIF
ENDONE
*****
* END OF DYNAMIC ALLOCATION JCL DD STATEMENTS.

FILE ARFILE FB 352
COPY ARFIELDS
XMLFILE XMLOUT FIELDS ACCOUNT NAME ADD1 BALANCE
XMLWRITE XMLOUT
/*
//
```

Figure 391 JCL to Dynamically Allocate Files Using HFS Path (Page 2 of 2)

The HFS path names in the sample program shown in [Figure 391](#) must be enclosed within single quotation marks. Double quotation marks are used to define the entire JCL line. You must use the extensions xml and xsd for the XML data and schema documents, respectively. When the XMLXSL DD statement is present, VISION:Results automatically allocates an HFS path name similar to the HFS path name but with an XSL ending qualifier. For example, if the HFS path name for the data document is

mydata.xml

then the HFS path name for the style sheet would be as follows:

/your/XML/path/mydata.xml.xsl

When the XML API allocates the HFS path name for a style sheet, it internally specifies a DDNAME similar to the data document, ending with an L. The style sheet DDNAME must not be coded in the JCL or a style sheet will not be created.

The JCL required for Dynamic Allocation can be read and moved into working storage, as demonstrated in [Example 2—Allocate Facility and MOVE Command in OUTJCL Fields on page 321](#).

Dynamic Allocation of XML Documents with PDSE

The following is an example of how to dynamically create XML documents using PDSE.

```
//XMLFILE EXEC PGM=DYL280,REGION=0M
//STEPLIB DD DISP=SHR,DSN=YOUR.RESULTS.LOAD.LIBRARY
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(4,4))
//SYSPRINT DD SYSOUT=*
//SYS280R DD SYSOUT=*
//ARFILE DD DISP=SHR,DSN=YOUR.RESULTS.DEMOFILE
//SYSCOPY DD DISP=SHR,DSN=YOUR.RESULTS.SOURCE.LIBRARY
//***** XML STYLESHEET *****
//XMLXSL DD DISP=SHR,DSN=YOUR.RESULTS.HTMLLIB(XSL1)
//*
//SYSIN DD *
*****
* DYNAMIC ALLOCATION JCL DD STATEMENTS ARE BELOW.
*****
WORKAREA
  JCLSTAT 1
  MSG1 41 VALUE 'SYNTAX ERROR IN DYNAMICALLY ALLOCATED'
        'JCL'
  MSG2 47 VALUE 'ALLOCATION FAILURE IN DYNAMICALLY'
        'ALLOCATED JCL'
WORKAREA
  OUTJCL          160 1
  FILLER          80 1 VALUE
  "//XMLEOUT DD DSN=YOUR.XML.PDSE(data),DISP=SHR"
  FILLER          80  VALUE
  "//XMLEOUTS DD DSN=YOUR.XML.PDSE(schema),DISP=SHR"

ON ONE

ALLOCATE OUTJCL STATUSFLAG JCLSTAT
  IF JCLSTAT EQ 'N'
    PRINT MSG1
    STOP
  ENDIF
  IF JCLSTAT EQ 'F'
    PRINT MSG2
    STOP
  ENDIF
ENDONE
*****
* END OF DYNAMIC ALLOCATION JCL DD STATEMENTS.

FILE ARFILE FB 352
COPY ARFIELDS
XMLFILE XMLEOUT FIELDS ACCOUNT NAME ADD1 BALANCE
XMLWRITE XMLEOUT
/*
//
```

Figure 392 Example of how to Dynamically Allocate an XML Documents Using a PDSE Library

When the XMLXSL DD statement is present, VISION:Results dynamically allocates a PDSE member name for the style sheet in the same PDSE as the data document. The style sheet member name will be similar to the member name used for the data document, but ending with an L. For instance, if the PDSE member name for the data document is

MYDATA

then the PDSE member name for the style sheet would be as follows:

YOUR.XML.PDSE(MYDATA~~L~~)

Note: If the length of the XML member name is greater than seven characters, then the last character of the XSL name is replaced with an L.

When the XML API allocates the PDSE member name for a style sheet, it internally specifies a DDNAME similar to the data document, ending with an L. The style sheet DDNAME must not be coded in the JCL or a style sheet will not be created.

The JCL required for Dynamic Allocation can be read and moved into working storage, as demonstrated in [Example 2—Allocate Facility and MOVE Command in OUTJCL Fields on page 321](#).

Dynamic Allocation Using Sequential Data Sets

The following is an example of how to dynamically allocate XML documents using sequential data sets.

```
//XMLFILE EXEC PGM=DYL280,REGION=0M
//STEPLIB DD DISP=SHR,DSN=YOUR.RESULTS.LOAD.LIBRARY
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(4,4))
//SYSPRINT DD SYSOUT=*
//SYS280R DD SYSOUT=*
//ARFILE DD DISP=SHR,DSN=YOUR.RESULTS.DEMOFILE
//SYSCOPY DD DISP=SHR,DSN=YOUR.RESULTS.SOURCE.LIBRARY
//SYSIN DD *
*****
* DYNAMIC ALLOCATION JCL DD STATEMENTS ARE BELOW.
*****
WORKAREA
  JCLSTAT 1
  MSG1 41 VALUE 'SYNTAX ERROR IN DYNAMICALLY ALLOCATED'
        'JCL'
  MSG2 47 VALUE 'ALLOCATION FAILURE IN DYNAMICALLY'
        'ALLOCATED JCL'
WORKAREA
      OUTJCL          480 1
      FILLER          80 1 VALUE
  "//XMLEOUT DD DSN=YOUR.XML.DATA,DISP=(,CATLG), "
      FILLER          80 VALUE
  "// UNIT=SYSDA,SPACE=(TRK,(15,15)), "
      FILLER          80 VALUE
  "// DCB=RECFM=VB,LRECL=8024,BLKSIZE=8028) "
      FILLER          80 VALUE
  "//XMLEOUTS DD DSN=YOUR.XML.SCHEMA,DISP=(,CATLG), "
      FILLER          80 VALUE
  "// UNIT=SYSDA,SPACE=(TRK,(15,15)), "
      FILLER          80 VALUE
  "// DCB=RECFM=VB,LRECL=8024,BLKSIZE=8028) "

ON ONE

ALLOCATE OUTJCL STATUSFLAG JCLSTAT
  IF JCLSTAT EQ 'N'
    PRINT MSG1
    STOP
  ENDIF
  IF JCLSTAT EQ 'F'
    PRINT MSG2
    STOP
  ENDIF
ENDONE
```

Figure 393 Example of how to Dynamically Allocate XML Documents Using Sequential Data Sets (Page 1 of 2)

```
*****  
* END OF DYNAMIC ALLOCATION JCL DD STATEMENTS.  
  
FILE ARFILE FB 352  
COPY ARFIELDS  
XMLFILE XMLOUT FIELDS ACCOUNT NAME ADD1 BALANCE  
XMLWRITE XMLOUT  
/*  
//
```

Figure 393 Example of how to Dynamically Allocate XML Documents Using Sequential Data Sets (Page 2 of 2)

VISION:Results does not create a style sheet when the XML data and schema documents are written to sequential data sets. Instead, the web browser default style sheet is used to format and view the document.

The JCL required for dynamic allocation can be read and moved into working storage, as shown in [Example 2—Allocate Facility and MOVE Command in OUTJCL Fields on page 321](#).

XML Data and Schema Document Generation

This section describes the data and schema documents generated by the VISION:Results XMLFILE and XMLWRITE statements.

Sample Program

The following sample program produces XML data and schema documents, such as those shown in [Figure 395](#), [Figure 396](#), and [Figure 397](#).

Note: [Figure 394](#) is not a complete VISION:Results program.

```
FILE ARFILE FB 352  
COPY ARFIELDS  
XMLFILE XMLOUT FIELDS ACCOUNT NAME ADD1 BALANCE  
XMLWRITE XMLOUT
```

Figure 394 Sample Program to Produce XML Data and Schema Documents

XML Data Document Without Style Sheet

The sample program ([Figure 394](#)) produces the following data document when a style sheet is not used. A style sheet is not used when the XMLXSL is not coded in the JCL or the XML documents are allocated to sequential data sets.

```
<?xml version="1.0" ?>
<XMLOUT-XML-DOCUMENT>

  <XML-TABLE-ROW>
    <ACCOUNT>8006547</ACCOUNT>
    <NAME>TORRES, ERNESTO</NAME>
    <ADD1>23444 PARK LANE</ADD1>
    <BALANCE> 44.99</BALANCE>
  </XML-TABLE-ROW>

  <XML-TABLE-ROW>
    <ACCOUNT>6208657</ACCOUNT>
    <NAME>CHO PYUNG, SUH</NAME>
    <ADD1>33333 PALL MALL</ADD1>
    <BALANCE> 32.00</BALANCE>
  </XML-TABLE-ROW>

  .
  .
  .
  .

</XMLOUT-XML-DOCUMENT>
```

Figure 395 XML Data Document Produced Without a Style Sheet

XML Data Document with Style Sheet

The sample program ([Figure 394](#)) produces the following data document when a style sheet is used.

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl"
  href="YOUR.XML.PDSE(MYDATA1)" ?>

<XMLOUT-XML-DOCUMENT>
  <XML-TABLE-ROW>
    <ACCOUNT>8006547</ACCOUNT>
    <NAME>TORRES, ERNESTO</NAME>
    <ADD1>23444 PARK LANE</ADD1>
    <BALANCE> 44.99</BALANCE>
  </XML-TABLE-ROW>

  <XML-TABLE-ROW>
    <ACCOUNT>6208657</ACCOUNT>
    <NAME>CHO PYUNG, SUH</NAME>
    <ADD1>33333 PALL MALL</ADD1>
    <BALANCE> 32.00</BALANCE>
  </XML-TABLE-ROW>

  .
  .
  .
  .

</XMLOUT-XML-DOCUMENT>
```

Figure 396 XML Data Document Produced With a Style Sheet

XML Schema Document

The sample program ([Figure 394](#)) produces the following schema document. Only one schema document is produced, whether a style sheet is used or not.

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="XMLEOUT-XML-DOCUMENT">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">

        <xs:element name="XML-TABLE-ROW"
          type="rowType"/>

      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="rowType">
    <xs:sequence>

      <xs:element name="ACCOUNT"
        type="fieldType1" minOccurs="0"/>
      <xs:element name="NAME"
        type="fieldType2" minOccurs="0"/>
      <xs:element name="ADD1"
        type="fieldType3" minOccurs="0"/>
      <xs:element name="BALANCE"
        type="fieldType4" minOccurs="0"/>

    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="fieldType1">
    <xs:restriction base="xs:string">
      <xs:maxLength value="7"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="fieldType2">
    <xs:restriction base="xs:string">
      <xs:maxLength value="25"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="fieldType3">
    <xs:restriction base="xs:string">
      <xs:maxLength value="25"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="fieldType4">
    <xs:restriction base="xs:decimal">
      <xs:fractionDigits value="2"/>
      <xs:totalDigits value="9"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

Figure 397 XML Schema Document

XML Documents on PC Web Browser

This section shows the final documents that are displayed on your PC's web browser.

eXtensible Stylesheet Language Transformation (XSLT)

XSLT is a language that is designed to convert or transform one XML document into another XML document. It is also used to convert XML into HTML. XSLT is used to transform the structure and content of an XML document. An XSLT language processor is embedded or included in most modern web browsers and is used to display an XML document by converting it into HTML. The XML document is linked to an XSL document which describes how to perform the transformation. The web browser executes the XSL instructions and displays the result of the transformation, which is a properly formatted and displayable HTML document.

Sample XSL Style Sheets

Two XSL sample style sheets are provided in the VISION:Results HTMLLIB library: XSL and XSL1. The XSL style sheet creates a simple composite HTML table, as shown in [Figure 398](#).

XMLOUT-XML-DOCUMENT			
ACCOUNT	NAME	ADD1	BALANCE
8006547	TORRES,ERNESTO	23444 PARK LANE	44.99
6208657	CHO PYUNG,SUH	33333 PALL MALL	32.00
7082509	S.F.MEM.HOSP.	6789 OLD MAN RD	5.00
6002587	SANTA FE HOSP ASSN	1212 WISCONSIN DRIVE	15.00
6107265	GENVARDI,G J	9785 FLEET STR	43.00
6095631	TODIPE,MICHAEL	5678 REGENT SQR	45.24
6123228	SILVA,JULIAN	33356 WIZZY DR #32	0.00
6059708	CHAVEZ,RAY	7986 MAYFLOWER AVE	15.00
6044395	CANO,MICHEAL S	56479 BOND STREET #12	15.00
6024963	HILL,GARY E	5445 COVENTRY ST	3.80
8011508	HUGHES,RAY	3025 WINTER ST	178.70

Figure 398 Sample Data Document Using XSL Style Sheet

XSL1 style sheet contains advanced HTML features composed of several frames where the headers are separated from the data, as shown in [Figure 399](#)

Xmldata Xml Document			
Account	Name	Addr	Balance
8006547	TORRES,ERNESTO	23444 PARK LANE	44.99
6208657	CHO PYUNG,SUH	33333 PALL MALL	32.00
7082509	S.F.MEM.HOSP.	6789 OLD MAN RD	5.00
6002587	SANTA FE HOSP ASSN	1212 WISCONSIN DRIVE	15.00
6107265	GENVARDI,G J	9785 FLEET STR	43.00
6095631	TODIPE,MICHAEL	5678 REGENT SQR	45.24
6123228	SILVA,JULIAN	33356 WIZZY DR #32	0.00
6059708	CHAVEZ,RAY	7986 MAYFLOWER AVE	15.00
6044395	CANO,MICHEAL S	56479 BOND STREET #12	15.00
6024963	HILL,GARY E	5445 COVENTRY ST	3.80

Figure 399 Sample Data Document Using XSL1 Style Sheet

The differences between the data documents created by the two style sheets are subtle in these fixed images, but you can see the text in the header and column headings is all upper case and the separators are visible in the XSL document. The text is mixed case with clean spacing between words in the XSL1 document. The most significant difference in the XSL1 document is the frame construction, which enables the headers to remain stationary while the data portion of the table can be scrolled.

If your web browser does not support HTML features, then the XSL style sheet should be used.

XML Data Document Without a Style Sheet

The following is the XML data document produced without a style sheet.

```
<?xml version="1.0" ?>
- <XMLOUT-XML-DOCUMENT>
- <XML-TABLE-ROW>
  <ACCOUNT>8006547</ACCOUNT>
  <NAME>TORRES,ERNESTO</NAME>
  <ADD1>23444 PARK LANE</ADD1>
  <BALANCE>44.99</BALANCE>
</XML-TABLE-ROW>
- <XML-TABLE-ROW>
  <ACCOUNT>6208657</ACCOUNT>
  <NAME>CHO PYUNG,SUH</NAME>
  <ADD1>33333 PALL MALL</ADD1>
  <BALANCE>32.00</BALANCE>
</XML-TABLE-ROW>
- <XML-TABLE-ROW>
  <ACCOUNT>7082509</ACCOUNT>
  <NAME>S.F.MEM.HOSP.</NAME>
  <ADD1>6789 OLD MAN RD</ADD1>
  <BALANCE>5.00</BALANCE>
</XML-TABLE-ROW>
- <XML-TABLE-ROW>
  <ACCOUNT>6002587</ACCOUNT>
  <NAME>SANTA FE HOSP ASSN</NAME>
  <ADD1>1212 WISCONSIN DRIVE</ADD1>
  <BALANCE>15.00</BALANCE>
```

Figure 400 XML Data Document Produced Without a Style Sheet

The default Internet Explorer web browser is being used to format the data. This is useful when exporting to an application that uses the data as input, such as Microsoft Excel. The XSL style sheet also can be used to export data directly into Microsoft Excel, but the XSL1 style sheet does not support this function because of its use of advanced HTML/JavaScript techniques.

XMLXSL ddname

The XSL template is enabled by adding the XMLXSL ddname. The ddname points to one of the previously mentioned templates, or a customized version. When the XMLXSL ddname is present, all XML files that are created are linked to a customized copy of the template.

This capability is enabled only for PDSE and HFS XML files:

- PDSE: The XSL template is copied to a member name consisting of the XML member name with an L appended to the end in the same PDSE library as the XML file. If the length of the XML member name is greater than seven characters, the last byte of the XSL name is replaced with an L.
- HFS: The XSL template is copied to an HFS file consisting of the XML HFS file name appended with .xsl in the same directory as the XML file.

XML HTML Integration

When HTML and XML files are created together, links to the XML files are created and added to the HTML file Table Of Contents page. This feature requires that:

- The XMLXSL ddname feature is enabled.
- The XML and HTML files reside in the same type of file system.
- Either all files are HFS, or all files are PDSE.
- The HTML file is the file that was initially accessed by the web browser.

An example is shown in [Figure 401](#).

Report Contents	ACCOUNT NUMBER	NAME	STREET	BALANCE
• Page 1				
XML Report Tables				
• XMLOUT-XML-DOCUMENT	8006547	TORRES, ERNESTO	23444 PARK LANE	44.99
	8208657	CHO PYUNG, SUH	33333 FALL MALL	32.00
	7082509	S. F. MEM. HOSP.	6789 OLD MAN RD	5.00
	6002587	SANTA FE HOSP ASSN	1212 WISCONSIN DRIVE	15.00
	6107265	GENUARDI, G J	9785 FLEET STR	49.00
	6095691	TODIFE, MICHAEL	5678 REGENT SQ	45.24
	6123228	SILVA, JULIAN	33356 WISZY DR #32	.00
	6059708	CHAVEZ, RAY	7986 MAYFLOWER AVE	15.00
	6044395	CANO, MICHAEL S	56479 BOND STREET #12	15.00
	6024963	HILL, GARY E	5445 COVENTRY ST	3.80
	8011508	HUGHES, RAY	3025 WINTER ST	178.70
	2002299	PLACIDO, ORTEGA	1247 S ST LOUIS ST	413.58
	6123317	VASQUEZ, IRENE	229 W 7TH ST	.00
	6112536	CHAVEZ, NORMA A	2483 BUCKTOWN	55.00
	6009166	LOCKE, JEFFREY	321 PALMOROV AVE	15.00
	6012644	MONTES, CARMEN	1556 GLADYS	89.28
	6218113	AGUIERA, EMILIO	4922 W HONEYWOOD CT	108.44
	6004695	SANTA FE RR ASSN	532 WILLMONT	32.02
	6016316	VANCE, VERNON	717 W ARLETTE	40.76
	2005131	WICINSKI, ALEXANDER B	610 SOLWAY ST	848.71
	6080669	WILLIAMS, JAMES	610 SO. ST. LOUIS S	24.24
	6088678	TYLER, JAMES	610 POPLAR ST	305.64
	1014621	WOOD, ROGER D	123 MONTANA	1,702.42
	6042325	CHAMBERLAIN, FRANCES	10243 SANTA ANA BL	34.24
	6041272	MARTIN, GAYLORD L	2327 1/2 OFAL	15.00
	6049354	SWARTZ, GEORGE	3442 ASTOR AVE	21.25
	6067956	PORTER, MAY T	4821 ROJAS ST	80.24
	6062946	BROCK, ETHEL G	15544 TWIN	77.00
	6103375	GARKOW, DOROTHY	3423 ROSEVIEW AVE	25.00
	6101291	RACH, MARY	2660 LADD AVE	35.00

Figure 401 Sample Table Showing XML and HTML Integration

XML Data Document with a Link to an HTML Report

This section shows how XML data documents can be linked to HTML reports.

Sample Program

The following sample program creates a hyperlink to an HTML report.

```
FILE ARFILE FB 352
COPY ARFIELDS

REPORT 80 WIDE HTML HTMLOUT COUNT 500 STYLE 00

XMLFILE XMLOUT FIELDS ACCOUNT NAME ADD1 BALANCE
XMLWRITE XMLOUT

LIST ACCOUNT NAME ADD1 BALANCE
```

Figure 402 Sample Program to Link XML Data Document to HTML Report

The following XML data document is produced by the sample program.

Report Contents	ACCOUNT NUMBER	NAME	STREET	BALANCE
• Page 1	8006547	TORRES,ERNESTO	23444 PARK LANE	44.99
XML Report Tables	6208657	CHO PYUNG,SUH	33333 PALL MALL	32.00
• XMLOUT-XML-DOCUMENT	7082509	S.F.MEM.HOSP.	6789 OLD MAN RD	5.00
	6002587	SANTA FE HOSP ASSN	1212 WISCONSIN DRIVE	15.00
	6107265	GENVARDI,G J	9785 FLEET STR	43.00
	6095631	TODIPE,MICHAEL	5678 REGENT SQR	45.24
	6123228	SILVA,JULIAN	33356 WIZZY DR #32	.00
	6059708	CHAVEZ,RAY	7986 MAYFLOWER AVE	15.00
	6044395	CANO,MICHAEL S	56479 BOND STREET #12	15.00
	6024963	HILL,GARY E	5445 COVENTRY ST	3.80
	8011508	HUGHES,RAY	3025 WINTER ST	178.70
	2002299	PLACIDO,ORTEGA	1247 S ST LOUIS ST	413.58
	6123317	VASQUEZ,IRENE	329 W 7TH ST	.00
	6112596	CHAVEZ,NORMA A	2489 EUCOTOWN	55.00
	6009166	LOCKE,JEFFREY	321 PALMOROV AVE	15.00
	8012644	MONTES,CARMEN	1556 GLADYS	89.28
	6218113	AGUIERA,EMILIO	4922 W HONEYWOOD CT	108.44
	6004695	SANTA FE RR ASSN	532 WILLMONTE	32.02
	6016316	VANCE,VERNON	717 W ARLETE	40.76
	2005131	WICINSKI,ALEXANDER B	610 SOLWAY ST	848.71
	6080669	WILLIAMS,JAMES	610 SO. ST. LOUIS S	24.24
	6088678	TYLER,JAMES	610 POPLAR ST	305.64
	1014021	WOOD, ROGER D	123 MONTARA	1,702.42
	6042325	CHAMBERLINE,FRANCES	10249 SANTA ANA BL	34.24
	6041672	HARTIN,GAYLORD L	2327 1/2 OPAL	15.00
	6049354	SWARTZ,GEORGE	3442 ASTOR AVE	21.25
	6067956	PORTER,MAY T	4821 ROJAS ST	80.24
	6062946	BROCK,ETHEL G	15544 TWIN	77.00
	6103375	GARKOU,DOROTHY	3423 ROSEVIEW AVE	25.00
	6101291	RACH,MARY	2660 LADD AVE	35.00

Figure 403 XML Data Document With Link to an HTML Report

When you click on XMLOUT-XML-DOCUMENT in the left column (circled in [Figure 403](#)), the web browser displays the XML document shown in the following figure.

Report Contents	ACCOUNT NUMBER	NAME	STREET	BALANCE
• Page 1	Xmlout Xml Document			
XML Report Tables	Account	Name	Addl	Balance
• XMLOUT-XML-DOCUMENT	8006547	TORRES,ERNESTO	23444 PARK LANE	44.99
	6208657	CHO PYUNG,SUH	33333 PALL MALL	32.00
	7082509	S.F.MEM.HOSP.	6789 OLD MAN RD	5.00
	6002587	SANTA FE HOSP ASSN	1212 WISCONSIN DRIVE	15.00
	6107265	GENVARDI,G J	9785 FLEET STR	43.00
	6095631	TODIPE,MICHAEL	5678 REGENT SQR	45.24
	6123228	SILVA,JULIAN	33356 WIZZY DR #32	0.00
	6059708	CHAVEZ,RAY	7986 MAYFLOWER AVE	15.00

Figure 404 Linked XML Document

From here, you can click on Page 1 in the upper left column (circled in [Figure 404](#)) to return to the HTML document.

Error Handling

VISION:Results default error handling entails printing a DYL-1240E error message, describing the error, and ending the program with a U800. If the STATUS keyword is present in the VISION:Results application code, then the user is responsible for the error handling. A value of N returned by the system in the dataname-s field indicates that an error occurred on the XMLWRITE. A value of blank indicates that the operation was successful. Coding the STATUS keyword applies to DYL-1240E errors only. Other errors are printed within the listings and cause the program to end with a U800 return code.

An XML API error consists of a completion code and a reason code. The completion code and reason code can be found in the DYLCOMRG reserved word. The completion code is in a 4-byte character format in bytes 3 through 6 of DYLCOMRG. The reason code is in a 4-byte character format in bytes 7 through 10 of DYLCOMRG. The first two bytes of DYLCOMRG are internally used and have no XML application.

WORKAREA Fields

Define the following WORKAREA fields in your program when overriding default error handling:

```
WORKAREA
XMLCODES      10
COMPCODE       4   3
REASCODE       4   7
```

Program logic may be added where the program checks the dataname-s field for an N. If an N is found, DYLCOMRG then can be moved into the XMLCODES field to automatically update the COMPCODE and REASCODE fields with the contents of the XML API's completion code and reason code, respectively. The fields then can be displayed. It is the user's responsibility to determine whether a return code value should be moved to DYLRETURN and whether to continue processing or terminate the program.

Following are two sample XML programs that generate a DYL-1240E error condition. Note the difference between default error handling ([Figure 405](#)) and user-controlled error handling ([Figure 406](#)).

```
COMPUTER ASSOCIATES      VISION:RESULTS  6.0 *
1-----VISION:RESULTS FREE FORM TEXT
FILE ARFILE FB 352
COPY ARFIELDS
COPIED ACCOUNT      7   4      (ACCOUNT'NUMBER)
COPIED ACCTNO       7   4      (ACCOUNT'NUMBER)
COPIED TRANDATE      6  38      (TRANSACTION'DATE)
COPIED TRANS        7   4      (TRANSACTION'NUMBER)
COPIED BILLNGDATE    6  44 NU D  (BILLING'DATE)
COPIED NAMADR       75  85      (NAME AND ADDRESS)
COPIED NAME         25  85
COPIED ADD1         25 110      (STREET)
COPIED ADD2         25 135      (STATE'ZIP)
COPIED BALANCE      5 170 PD 2 A
...
XMLFILE XMLOUT FIELDS ACCOUNT NAME ADD1 BALANCE
```

Figure 405 Sample Run With Default Error Handling

```

XMLWRITE XMLOUT

COMPUTER ASSOCIATES      VISION:RESULTS  6.0 *      DATE  10/11/05
19.05.27  PAGE          1
DYL-1240E  XML OPEN      ERROR-FILE XMLOUT  CODES AND MESSAGE FOLLOW: COMPLETION CODE =
0010,REASON CODE = 0036
          MAXIMUM RECORD LENGTH HAS BEEN EXCEEDED. SEE REASON CODE FOR MAX LENGTH ALLOWED

COMPUTER ASSOCIATES      VISION:RESULTS  6.0 *      DATE  10/11/05
19.05.27  PAGE          2
          VISION:RESULTS CONTROL TOTALS
          FILE          RECORD          CHARACTER    BLOCK    DROPPED    REWRITTEN    INSERTED
ERASED
          ID          COUNT          COUNT          COUNT    BLOCK COUNT  RECORD COUNT  RECORD
COUNT RECORD COUNT
          ARFILE          1          352          PAGES
          2
          FILE PRINT
          REPORT PRINT
          FIXED BLANK COUNT
          FIXED DECIMAL DIVIDE
          COMPLETION CODE-0800          *** ERROR OCCURRED DURING THIS RUN ***

```

Figure 405 Sample Run With Default Error Handling

```

COMPUTER ASSOCIATES      VISION:RESULTS  6.0 *
1-----VISION:RESULTS FREE FORM TEXT-----
FILE ARFILE FB 352
COPY ARFIELDS
COPIED ACCOUNT      7  4      (ACCOUNT'NUMBER)
COPIED ACCTNO      7  4      (ACCOUNT'NUMBER)
COPIED TRANDATE      6 38      (TRANSACTION'DATE)
COPIED TRANS      7  4      (TRANSACTION'NUMBER)
COPIED BILLNGDATE    6 44 NU D (BILLING'DATE)
COPIED NAMADR      75 85      (NAME AND ADDRESS)
COPIED NAME      25 85
COPIED ADD1      25 110      (STREET)
COPIED ADD2      25 135      (STATE'ZIP)
COPIED BALANCE      5 170 PD 2 A
...
WORKAREA
XMLCODES 10
COMPCODE 4 3
REASCODE 4 7

XMLFILE XMLOUT FIELDS ACCOUNT NAME ADD1 BALANCE STATUS XMSTAT
XMLWRITE XMLOUT

IF XMSTAT EQ 'N'
MOVE DYLCOMRG TO XMLCODES
PRINT 'XML COMPLETION CODE = ' COMPCODE
PRINT 'XML REASON CODE = ' REASCODE
MOVE 12 TO DYLRETURN
STOP
ENDIF

COMPUTER ASSOCIATES      VISION:RESULTS  6.0 *      DATE  10/11/05
19.15.39  PAGE          1
          XML COMPLETION CODE = 0010
          XML REASON CODE = 0036

COMPUTER ASSOCIATES      VISION:RESULTS  6.0 *      DATE  10/11/05
19.15.39  PAGE          2
          VISION:RESULTS CONTROL TOTALS

```

Figure 406 Sample Run With User-Controlled Error Handling


```

      FILE      RECORD      CHARACTER      BLOCK      DROPPED      REWRITTEN
INSERTED  ERASED
      ID      COUNT      COUNT      COUNT      BLOCK COUNT      RECORD COUNT      RECORD
COUNT RECORD COUNT
      ARFILE      1      352
      RECORDS      2      PAGES      2
      FILE PRINT
      REPORT PRINT
      FIXED BLANK COUNT
      FIXED DECIMAL DIVIDE
      RETURN CODE-0012

```

Figure 406 Sample Run With User-Controlled Error Handling

XML API Completion and Reason Codes

This section describes the XML API completion and reason codes.

HCSDATA Module Codes

Message	Completion and Reason Codes	Meaning
Invalid Parameter: (internal)	Completion Code = 1 Reason Code = (n)	When (n) = 0, the Completion Code structure address in parameter 3 is invalid and the Completion Code can be retrieved from register #15 only. (This assumes the structure was set to 0 before the function call.) When (n) > 0, n = the parameter # of the invalid parameter field starting with #1. The invalid parameter could be a bad address, it could also be a value that is inconsistent with the purposes for which it is to be used. For example: ■ A field-count parameter should always be greater than zero. ■ The function code parameter #1 should always be a value between 1 and 10.
Invalid or incompatible translation requirements: (internal/external)	Completion Code = 2 Reason Code = 0	The Code Page translation specification on one of the output files is invalid.
The Open File Request Failed: (external)	Completion Code = 3 Reason Code = 1 or 2	The reason code represents which file handle was the cause of the failure.

Message	Completion and Reason Codes	Meaning
Field Index Array index value is illegal or out of range: (internal)	Completion Code = 4 Reason Code = (n)	The Field Index Array values must be greater than or equal to 0 and less than the maximum field-count of the Host Field Description Table. The reason code (n) represents the number of the field in the Field Index Array where the index value is incorrect.
Field Index Array index value is invalid: (internal)	Completion Code = 5 Reason Code = (n)	The values in the Field Index Array must be ascending. They do not have to be sequential. For example, the values could be 1, 2, 4, and 6. The reason code (n) represents the number of the field in the Field Index Array where the index value is incorrect.
Bad Zoned digit in the field data: (internal/external)	Completion Code = 6 Reason Code = (n)	The reason code (n) represents the number of the field in the parameter field list where the incorrect data is located. For example, if 4 fields are referenced in the field list, and field 3 is in error, then n=3.
Bad Packed digit in the field data: (internal/external)	Completion Code = 7 Reason Code = (n)	The reason code (n) represents the number of the field in the parameter field list where the incorrect data is located. For example, if 4 fields are referenced in the field list, and field 3 is in error, then n=3.
Low on memory: (external)	Completion Code = 8 Reason Code = (n)	
Binary File Translation and file format are incompatible: (internal/external)	Completion Code = 9 Reason Code = 0	When the file translation requested is a non-EBCDIC code page, the output files are opened and written in binary mode. The file format must not be fixed block, ASA, or machine character format.
The maximum record length has been exceeded: (internal/external)	Completion Code = 10 Reason Code = maximum record length allowed.	When the output file is a non-binary MVS data set, record truncation is an error and is not allowed.

Message	Completion and Reason Codes	Meaning
File Write Output Error: (internal/external)	Completion Code = 11 Reason Code = C++ Run-time library error code.	The C++ run-time library has encountered an error writing to the output file.
Character Translation Error: (internal/external)	Completion Code = 12 Reason Code = (n)	The reason code can be the following: ■ n=1—The translation descriptor is invalid. This is an internal error and should not occur. Contact Technical Support at http://ca.com/support. ■ n=2—A sequence of characters is not valid for the input code page. ■ n=3—A partial sequence of characters, shift-in without a shift-out, which is invalid has been detected.
Character Translation Warning: (internal/external)	Completion Code = 13 Reason Code = (n)	There were (n) numbers of characters in the input code page that were valid for the input code page, but did not have a corresponding valid translation in the output code page.
Bad Parameter Memory Address: (internal)	Completion Code = 14 Reason Code = (n)	An invalid parameter memory address has been provided. The value (n) is the parameter number of the address in error.
Field Name Length is greater than the maximum allowed: (internal)	Completion Code = 15 Reason Code = (n)	The maximum field name length is currently (64). The value (n) is the field number of the field in error in the Host Field Description Table starting from #0.
PCFILE Schema Creation error: (internal)	Completion Code = 16 Reason Code = (n)	The reason code can be the following: ■ n=2—bad field type ■ n=3—bad binary field length ■ n=4—bad binary field scale ■ n=5—bad character field length ■ n=6—bad packed field length ■ n=7—bad packed field scale ■ n=8—bad zoned field length ■ n=9—bad zoned field scale

Message	Completion and Reason Codes	Meaning
XML Schema Element not found: (internal)	Completion Code = 17 Reason Code = (n)	This error is returned by function code 5. The reason code can represent the following error conditions: ■ n=0—Schema Element Name not found. ■ n=1—Schema Element Index not found. ■ n=2—The element (previous/next/parent/child) does not exist. ■ n=3—The element requested does not exist. This is an internal error and should not occur. Contact Technical Support at http://ca.com/support.
XML Schema Parsing Error: (internal)	Completion Code = 18 Reason Code = (n)	This error is returned by function code 4. The reason code can represent the following error conditions: ■ n=1—Parser initialization error. Contact Technical Support at http://ca.com/support. ■ n=2—Error during parsing. ■ n=3—DOM error during parsing. ■ n=4—Unexpected error during parsing. ■ n=5—Errors occurred; no output available.
XML Schema Element Field Name not found: (internal)	Completion Code = 19 Reason Code = (n)	This error is returned by function code 6. A field element name in the Host Field Description Table is not found in the schema. The reason code is the number of the field not found in the table starting from #0.

Chapter 18: Multiple Reports and Multiple Requests

VISION:Results can produce up to 999 reports and requests in a single execution, with each report in any sequence you choose. Multiple-report programs can be frozen and restored in the same manner as single-report VISION:Results programs. You also can specify that processing is restarted at a particular report.

Each report request is coded in the same way that a single request is coded. You have the same selection, manipulation, calculation, and reporting capability in each request that is available in a single request. For information on single reports, see [Chapter 16: Using Report Statements](#). VISION:Results provides some additional commands to facilitate passing data from one report request to another.

The following statements deal specifically with producing multiple reports and requests in VISION:Results.

REPORTnnn Statement	(required)
USE Statement	(optional)
PICNSAVE Statement	(optional)

REPORTnnn Statement

The REPORTnnn command indicates the start of a new report request. The nnn is a 1- to 3-digit user identifier number. This number is checked to verify that it is greater than the previous report identifier, and it is then ignored. VISION:Results assigns its own report numbers, starting at 1.

The REPORT command for the first request is not required to have the identifier number, but the second through nth requests must have the 1- to 3-digit number. However, if you use an identifier number in the REPORT command for the first VISION:Results request, it must be either 1 or 0. If you use 0, the identifier number for subsequent reports must be 2 or greater.

Following the REPORT command, you can code any valid VISION:Results statements, including FILE, WORKAREA, and SORT, as well as reporting statements. You do not have to produce a report in all your VISION:Results requests. You must, however, specify the REPORTnnn command in each request, even if there is no report.

The format of the REPORTnnn statement is:

```
REPORTnnn [n WIDE] [n LONG] [n BETWEEN]
           [{module|name|ASA}] [PLUnnn] [SYS280Rx]
```

Figure 407 REPORTnnn Statement Format

SYS280Rx

Although the keyword SYS280Rx is for z/OS users only, VSE users can accomplish the same thing with the PLUnn keyword. If used, the x is any character valid in a ddname, including a blank. SYS280Rx indicates to which DD the report is to be written. Reports usually are written to SYS280R. If you are producing multiple reports that are to be printed on different forms, you probably want some reports written to a different DD. VISION:Results writes the report to the ddname specified in your REPORT statement.

A ddname (or logical unit, for VSE users) can have only one width, even if multiple reports with differing widths are to be printed. If the first report printed is not the widest, any reports that are wider are truncated to the width of the first report. With the SYS280Rx statement, you can use different ddnames for reports of different widths, removing any possibility of conflict.

For a description of all the remaining operands of the REPORTnnn statement, see [Report Statement on page 354](#).

USE Statement

The USE command indicates that information from a previous VISION:Results request is to be made available to this current request. In particular, file definitions and their associated data definitions can be carried forward from one request to another. Similarly, data that has been selected and saved using a PICNSAVE can be referenced in subsequent requests using the USE command. If file and field definitions are to be carried forward from one request to another, the original FILE statement must have specified RETAIN.

The format of the USE statement is:

```
USE idname
USE idname STATUS dataname
```

where idname is a file name that was retained in a previous request or is the ID of a PICNSAVE statement from a previous request.

If `idname` is a file name from a previous request, all data names defined with the original `FILE` statement are available. If you want to specify additional data names, this can be done following the `USE` statement. If the `FILE` in the original request is an output file, it is assumed to be input in the request where `USE` is specified. You can allow `VISION:Results` to read this file or you can specify `READ`. The same constraints and options apply as though the `FILE` were coded in this request. Totals for this file are printed on the control totals page of the current request.

If `idname` is the ID name from a `PICNSAVE` statement, the data as specified in the `PICNSAVE` is available in this request. The same data names and formats apply. Do not try to redefine this data following the `USE` statement. You can allow `VISION:Results` to read this data, or you can control the reading by issuing your own `READ` and specifying the ID name. The same constraints and options apply as though this were a `FILE` coded in this request. This includes sorting the ID. Totals are not printed for this ID on the control totals page. If there is not a `SORT` in the current request, you must supply a [STOP](#).

A file that is used in a secondary request cannot be specified as the operand of an `UNTIL FILE` in a `SORT` statement. For example:

```
USE ARFILE
SORT ARFILE USING ACCOUNT UNTIL ARFILE
```

The `SORT` statement is invalid because of the `UNTIL ARFILE` statement.

STATUS dataname

The keyword `STATUS` allows you to supply a data name for a status indicator set up by `VISION:Results`. The data name supplied, following the keyword, refers to the 1-byte indicator. When the file or saved data referred to by the `USE` is exhausted, an E is placed in this byte.

In the case, where the ID name specified in the `USE` statement refers to data that was saved using `PICNSAVE` by a prior request, it is your responsibility to terminate the request. If the request does not have a `SORT`, you must include a `STOP` statement in the request.

The best way to do this is to supply a `STATUS dataname` on the `USE` statement, then check this field for E and issue a `STOP` if the test is true. For example:

```
REPORT2
USE REP2DATA STATUS REP2IND
.
.
IF REP2IND EQ 'E' STOP ENDIF
.
.
```

Figure 408 Request Does Not Have a `SORT`

PICNSAVE Statement

The PICNSAVE command saves selected data to be passed to subsequent VISION:Results requests. This data can be selected data from a record, from work areas built by you, or from entire records. (The temporary output file created by PICNSAVE is fixed format, with the logical record length equal to the length of all the combined data names saved.) This saved data can then be referenced in a subsequent request using a USE command.

The format of the PICNSAVE statement is:

```
PICNSAVE idname USING dataname1 dataname2 ...  
      [BY nnnnn] [SYSnnn] [TAPE]  
      [DISK [{2311|2314|3330|3340|3350|3375|3380|FBA}]]
```

Figure 409 PICNSAVE Statement Format

idname

This is a 1- to 8-character (1- to 7-character for VSE) identifier associated with this saved data. When you want to reference this saved data in a subsequent VISION:Results request, specify this name. You can specify a particular ID name in a PICNSAVE statement only once per VISION:Results run, not per request. If you are saving data from various places in your VISION:Results request, you must perform a routine using a [PERFORM](#) statement that issues the PICNSAVE statement.

z/OS users must provide a DD statement, either in the JCL or using the ALLOCATE facility, with a ddname that is the same as the ID name. This DD statement specifies a data set (probably temporary) to hold the saved data.

For VSE users, this ID name is used in the DLBL or TLBL statement that describes the file (probably temporary) to hold the saved data.

It is your responsibility to provide JCL for each PICNSAVE ID name used in the VISION:Results program.

USING dataname

The keyword USING specifies the data to be saved. One or more data names are coded following the keyword USING. The length of the data and its attributes are determined by the individual data definitions. The format of the data is not changed when it is saved. The combined length of all fields being saved cannot be greater than 32767 bytes.

If you use a file name instead of a data name, a validation error occurs.

BY nnnnn

The keyword BY indicates a blocking factor to be used for the saved information. BY is followed by a 1- to 5-digit number indicating how many sets of saved data are to be grouped together. If BY is omitted, the saved data is written to a temporary file in unblocked format. For z/OS only, if the following apply, then the operating system calculates the block size:

- BY is omitted.
- The file is a non-VSAM output file.
- SYSBLOK was selected in the DYLINSTL macro.
- No JCL block size was specified.

The combined length of all the fields being saved multiplied by the blocking factor cannot exceed 32767.

If the sum of the lengths of all fields that are to be part of the PICNSAVE statement is less than 20, then the total length is rounded up to 20 because VISION:Results requires records to be at least 20 bytes in size. The new record length must be used in the JCL when describing the DCB; the true sum of the lengths is not valid in this case. This also means that the blocking factor can never be greater than 1638, even for the smallest record length, because $20 \times 1638 = 32760$ and the block length maximum is 32767. BY must not be specified in conjunction with DISK FBA.

SYSnnn

The keyword SYSnnn is used for VSE only. If coded, it indicates the SYS number for the temporary save area for the PICNSAVE data. If the data is to be saved on tape, SYSnnn is mandatory. The nnn must be three digits from 001 to 255.

DISK nnnn

The keyword DISK is used for VSE only. In VSE r2.1 and above, the nnnn is optional and is ignored if specified. If coded, it indicates that the PICNSAVE data is to be saved on disk.

TAPE

The keyword TAPE is used for VSE only. If coded, it indicates that the PICNSAVE data is to be saved on tape. If tape is specified as a standard labeled tape, the SYSnnn must also be specified. If neither TAPE nor DISK is specified, TAPE is assumed.

Multiple Requests

When the letter writing, linear regression, random or interval selection, scatter diagrams, and trend line functions are used in a secondary request of a multiple-request program, the defining statements LTH, LINEAR, SAMPLE, SCATTER, and TREND must follow immediately after the FILE and field definition statements and must precede the program procedure logic.

During execution, each function starts again within each unique request; the status of each function is not retained from request to request. For example, the selection functions in the program below are completely independent of each other. Two independent samples are chosen.

```
FILE ARFILE
  ACCOUNT 6

SAMPLE 01 RANDOM 5000 100
IF SAMPLING 01
  LIST ACCOUNT
ELSE REJECT ENDIF

REPORT2
USE ARFILE

SAMPLE 01 RANDOM 5000 100
IF SAMPLING 01
  LIST ACCOUNT
ELSE REJECT ENDIF
```

Figure 410 Two Examples of Functions Not Retained From Request to Request

Freezing and Restoring Multiple Reports

A multiple-report run can be frozen and restored by using the OPTION FREEZE and OPTION RESTORE commands. For information on the FREEZE and RESTORE features, see the *Advantage VISION:Results for z/OS Toolkit Reference Guide*.

Multiple-Report Examples

The following multiple-report examples are identical in the output that they produce. The approach taken for each is quite different. [Example 1 on page 459](#) uses the keyword RETAIN and the USE command to produce three reports. [Example 2 on page 464](#) uses two PICNSAVES (one PICNSAVE for the second report and a second PICNSAVE for the third report) and the USE command to produce the reports. The advantage of [Example 2 on page 464](#) over [Example 1 on page 459](#) is that only the selected data is saved and passed to the subsequent reports, not the entire file (PAYLAB). Both z/OS and VSE examples are provided.

z/OS

Note: In this section, the text being referred to in the callouts is in boldface.

Example 1

```

COMPUTER ASSOCIATES  VISION:RESULTS  5.0                      DATE  1/15/01
1-----VISION:RESULTS FREE  FORM-----72
*   MVS  EXAMPLE...USING THE KEYWORD RETAIN AND THE COMMAND USE
OPTION DATA ' 750 800 850 900 925 950 97510251150'
REPORT 80 WIDE SYS280RS
FILE PAYLAB FB 80 1600 RETAIN      ;< FILE STATEMENT
  DIVNO 2 (DIVN)
  DEPTNO 3 (DEPT)
  EMPLNO 5 (EMPL#)
  EMPLNAME 20 (NAME)
  HOURS 4 35 NU 1 E
  SHIFT 1 40 (SHF)
  RATE 4 45 NU 2 E
  RCODE 1 53
  PROJECTNO 5 11 (PROJECT'NUMBER)
  PHOURS 4 16 NU 1 E (HOURS'BILLED)
  RATESCHED 1 25 NU
WORKAREA
  PARMDATA 36
    RT1 4 1 NU 2 E
    RT2 4 NU 2 E
    RT3 4 NU 2 E
    RT4 4 NU 2 E
    RT5 4 NU 2 E
    RT6 4 NU 2 E
    RT7 4 NU 2 E
    RT8 4 NU 2 E
    RT9 4 NU 2 E
  AMOUNT 4 PD 2 E
  COMMENT 20 ( )
CONTROL DEPTNO DIVNO
  IF RCODE EQ 'L' REJECT  ENDIF
  AMOUNT = HOURS*RATE ROUNDED
  LIST SUPPRESS (DIVNO DEPTNO) EMPLNO EMPLNAME HOURS SHIFT
    RATE AMOUNT
ON CHANGE IN DEPTNO
  MOVE 'TOTAL DEPARTMENT' TO COMMENT
LISTBRK:
  LIST COMMENT AT DEPTNO SUM (HOURS AMOUNT) WITH 1 BEFORE 3 AFTER
ON CHANGE IN DIVNO
  MOVE '** DIVISION TOTAL **' TO COMMENT
  GOTO LISTBRK
ON FINAL
  MOVE '*** GRAND TOTAL ***' TO COMMENT
  GOTO LISTBRK
T1 'PAYROLL DETAIL BY DIVISION / DEPARTMENT'
T2 'PERIOD ENDING - '
T2+57 DYLDATE
REPORT2 80 WIDE
USE PAYLAB
WORKAREA

```

The REPORT statement indicates that the report is spooled to SYS280RS rather than SYS280R.

The keyword RETAIN passes the file attributes and field definitions to subsequent requests.

The REPORT2 statement is the beginning of the second request.

The USE statement indicates which file is processed in the request. The input field definitions are assumed.

Figure 411 z/OS Example 1 Using FILE RETAIN (Page 1 of 3)

```

COMPUTER ASSOCIATES VISION:RESULTS 5.0 DATE 1/15/01
1-----VISION:RESULTS FREE FORM-----72
  PARMDATA 36
    RT1 4 1 NU 2 E
    RT2 4 NU 2 E
    RT3 4 NU 2 E
    RT4 4 NU 2 E
    RT5 4 NU 2 E
    RT6 4 NU 2 E
    RT7 4 NU 2 E
    RT8 4 NU 2 E
    RT9 4 NU 2 E
  AMOUNT 4 PD 2 E
  COMMENT 20 ( )
CONTROL EMPLNO
  IF RCODE EQ 'L' REJECT ENDIF
  SORT PAYLAB USING EMPLNO SHIFT
  AMOUNT=HOURS*RATE ROUNDED
  LIST SUPPRESS EMPLNO (EMPLOYEE'NUMBER) SUPPRESS EMPLNAME
    DIVNO DEPTNO HOURS (HOURS'WORKED)
    SHIFT RATE (HOURLY'RATE) AMOUNT (GROSS'WAGES) ' '
ON CHANGE IN EMPLNO
  LIST SUM HOURS SUM AMOUNT '***' WITH 2 AFTER
ON FINAL
  LIST '*** FINAL TOTAL ***' AT EMPLNO SUM HOURS SUM AMOUNT '***'
  WITH 3 BEFORE
T1 'P A Y R O L L D E T A I L B Y E M P L O Y E E '
T2 'PERIOD ENDING - '
T2+70 DYLDATE
REPORT3 80 WIDE SYS280RS
OPTION PRINTEROR
USE PAYLAB ; <- USE STATEMENT
WORKAREA
  PARMDATA 36
    RT1 4 1 NU 2 E
    RT2 4 NU 2 E
    RT3 4 NU 2 E
    RT4 4 NU 2 E
    RT5 4 NU 2 E
    RT6 4 NU 2 E
    RT7 4 NU 2 E
    RT8 4 NU 2 E
    RT9 4 NU 2 E
  BILLED AMT 5 PD 3 E
  COMMENT 20 ( )
CONTROL PROJECTNO
ON ONE
  MOVE DYLPARM TO PARMDATA
ENDONE
  IF RCODE NE 'L' REJECT ENDIF
  SORT PAYLAB USING PROJECTNO EMPLNO

```

The REPORT3 statement is the beginning of the third report request.

The USE statement indicates which file is processed in the request.

Figure 411 z/OS Example 1 Using FILE RETAIN (Page 2 of 3)

```

COMPUTER ASSOCIATES VISION:RESULTS 5.0                      DATE 1/15/01
1-----VISION:RESULTS FREE  FORM-----72
      INX=(RATESCHED-1)*4
      BILLED_AMT=PHOURS*RT1(INX) ROUNDED
      LIST SUPPRESS (PROJECTNO EMPLNO) PHOURS RATESCHED BILLED_AMT ' '
ON CHANGE IN PROJECTNO
      LIST SUM (PHOURS BILLED_AMT) '***'
      WITH 1 BEFORE AND 3 AFTER
ON FINAL
      LIST SUM (PHOURS BILLED_AMT) '****'
      WITH 2 BEFORE
T1 'LABOR COST BY PROJECT FOR PERIOD ENDING-,
      WITH 2 AFTER
T2 '      RATE SCHEDULES: #1      #2      #3' FIXED
T3 '                        #4      #5      #6' FIXED
T4 '                        #7      #8      #9' FIXED
T1+70 DYLDATE
T2+24 RT1
T2+35 RT2
T2+46 RT3
T3+24 RT4
T3+35 RT5
T3+46 RT6
T4+24 RT7
T4+35 RT8
T4+46 RT9

```

Figure 411 z/OS Example 1 Using FILE RETAIN (Page 3 of 3)

PAYROLL DETAIL BY EMPLOYEE							01/15/01
EMPLOYEE NUMBER	NAME	DIVN	DEPT	PERIOD ENDING - HOURS WORKED	SHF	HOURLY RATE	GROSS WAGES
00009	MUMFORD, M.E.	01	432	40.0	1	8.50	330.00
		01	432	9.6	3	12.75	122.40
				49.6			452.40 **
01574	LORAC, M.B.	01	127	37.5	3	9.75	365.63
				37.5			365.63 **
03200	SQUIGLEY, GEO.D.	01	127	40.0	1	11.50	460.00
				40.0			460.00 **
04010	WASHINGTON, LEROY	02	952	32.0	2	9.75	312.00
				32.0			312.00 **
04012	SVENSEN, ERICA T.	02	952	40.0	2	9.50	380.00
				40.0			380.00 **
07123	ZAK, HERMAN P.	02	951	40.0	2	8.00	320.00
		02	951	6.0	3	12.00	72.00
				46.0			392.00 **
12370	PEREZ, JUAN PERICO	01	432	37.5	1	7.50	281.25
				37.5			281.25 **
23941	POOLE, M.T.	01	127	40.0	1	7.50	300.00
		01	127	10.5	2	11.25	118.13
				50.5			418.13 **
32785	JONES, ROBERT S.	02	951	39.0	1	10.25	399.75
				39.0			399.75 **
62134	GODFREY, Q.Z.	01	432	40.0	2	7.50	300.00
				40.0			300.00 **
65656	YELNATS, STANLEY	01	432	40.0	2	9.25	370.00
				40.0			370.00 **
74424	REEVES, SLADE	02	951	40.0	1	9.00	360.00
				40.0			360.00 **
77778	STIVIC, G.N.	02	952	40.0	2	8.50	340.00
				40.0			340.00 **
91101	WONG, LING SU	02	951	40.0	1	7.50	300.00
				40.0			300.00 **
97537	VON SCHMIDT, JOHANN	02	951	40.0	1	7.50	300.00
				40.0			300.00 **
** FINAL TOTAL **				612.1			5,431.16 **

Figure 412 Output from z/OS Example 1 (Page 1 of 2)

PAYROLL DETAIL BY DIVISION / DEPARTMENT							
				PERIOD ENDING -		01/15/01	
DIVN	DEPT	EMPL#	NAME	HOURS	SHF	RATE	AMOUNT
01	127	01574	LORAC, M.B.	37.5	3	9.75	365.63
		03200	SQUIGLEY, GEO.D.	40.0	1	11.50	460.00
		23941	POOLE, M.T.	40.0	1	7.50	300.00
		23941	POOLE, M.T.	10.5	2	11.25	118.13
TOTAL DEPARTMENT				128.0			1,243.76
01	432	00009	MUMFORD, M.E.	40.0	1	8.50	330.00
		00009	MUMFORD, M.E.	9.6	3	12.75	122.40
		12370	PEREZ, JUAN PERICO	37.5	1	7.50	281.25
		62134	GODFREY, Q.Z.	40.0	2	7.50	300.00
		65656	YELNATS, STANLEY	40.0	2	9.25	370.00
TOTAL DEPARTMENT				167.1			1,403.65
** DIVISION TOTAL **				295.1			2,647.41
02	951	07123	ZAK, HERMAN P.	40.0	2	8.00	320.00
		07123	ZAK, HERMAN P.	6.0	3	12.00	72.00
		32785	JONES, ROBERT S.	39.0	1	10.25	399.75
		74424	REEVES, SLADE	40.0	1	9.00	360.00
		91101	WONG, LING SU	40.0	1	7.50	300.00
		97537	VON SCHMIDT, JOHANN	40.0	1	7.50	300.00
TOTAL DEPARTMENT				205.0			1,751.75
02	952	04010	WASHINGTON, LEROY	32.0	2	9.75	312.00
		04012	SVENSEN, ERICA T.	40.0	2	9.50	380.00
		77778	STIVIC, G.N.	40.0	2	8.50	340.00
TOTAL DEPARTMENT				112.0			1,032.00
** DIVISION TOTAL **				317.0			2,783.75
*** GRAND TOTAL ***				612.1			5,431.16
RATE SCHEDULES:				FOR PERIOD ENDING -		01/15/01	
		#1 7.50	#2 8.00	#3 8.50			
		#4 9.00	#5 9.25	#6 9.50			
		#7 9.75	#8 10.25	#9 11.50			
PROJECT NUMBER	EMPL#	HOURS BILLED	RATESCHED	BILLED AMT			
19	01574	20.0	7	195.000			
	04010	28.9	7	281.780			
	07123	6.0	2	48.000			
	12370	30.0	1	225.000			
	23941	10.5	1	78.750			
	65656	12.6	5	116.550			
	91101	40.0	1	300.000			
		148.0		1,245.080			**
110	03200	40.0	9	460.000			
	04010	3.1	7	30.225			
	07123	32.0	2	256.000			
	23941	13.0	1	97.500			
	32785	20.0	8	205.000			
		108.1		1,048.725			**
162	00009	20.0	3	170.000			
	01574	17.5	7	170.625			
	65656	10.0	5	92.500			
		47.5		433.125			**
170	00009	20.0	3	170.000			
	65656	5.4	5	49.950			
	77778	17.0	3	144.500			
	97537	27.6	1	207.000			
		70.0		571.450			**
409	00009	9.6	3	81.600			
	12370	7.5	1	56.250			
	32785	19.0	8	194.750			
	65656	12.0	5	111.000			
	77778	19.2	3	163.200			
	97537	8.2	1	61.500			
		75.5		668.300			**
727	04012	40.0	6	380.000			
		40.0		380.000			**
998	07123	8.0	2	64.000			
	62134	40.0	1	300.000			
	74424	40.0	4	360.000			
	77778	3.8	3	32.300			
	97537	4.2	1	31.500			
		96.0		787.800			
				585.1	5,134.480	***	

Figure 412 Output from z/OS Example 1 (Page 2 of 2)

JCL for Example 1

```

//RESULTS JOB .....
//STEP01 EXEC PGM=DYL280
//STEPLIB DD DSN=your.DYL280.loadlib,DISP=SHR
//SYS280R DD SYSOUT=A
//SYS280RS DD SYSOUT=A,DCB=BLKSIZE=80
//SYSPRINT DD SYSOUT=A
//AUDPRINT DD SYSOUT=A
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//AUDWORK DD UNIT=SYSDA,SPACE=(TRK,(10,5))
//AUDEPF DD UNIT=SYSDA,SPACE=(TRK,(10,5)),
// DCB=(BLKSIZE=800,LRECL=80,RECFM=FB)
//AUDCBF DD UNIT=SYSDA,SPACE=(TRK,(10,5)),
// DD DCB=BLKSIZE=1000
//SYSOUT DD SYSOUT=A
//SORTLIB DD DSN=your.sortlib.dataset,DISP=SHR
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(5),,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(5),,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(5),,CONTIG)
//PAYLAB DD DSN=PAYROLL.LABEL,DISP=SHR
//SYSIN DD *
                                         Example 1 VISION:Results program
//

```

Figure 413 z/OS JCL for [Example 1](#)

Example 2

```

COMPUTER ASSOCIATES VISION:RESULTS 5.0                                DATE 1/15/01
1-----VISION:RESULTS FREE FORM-----72
* MVS EXAMPLE...USING PICNSAVE AND USE COMMANDS
OPTION DATA ' 750 800 850 900 925 950 97510251150'
REPORT 80 WIDE SYS280RS
FILE PAYLAB FB 80 1600 ;<--FILE STATEMENT
  DIVNO 2 (DIVN)
  DEPTNO 3 (DEPT)
  EMPLNO 5 (EMPL#)
  EMPLNAME 20 (NAME)
  HOURS 4 35 NU 1 E
  SHIFT 1 40 (SHF)
  RATE 4 45 NU 2 E
  RCODE 1 53
  PROJECTNO 5 11 (PROJECT'NUMBER)
  PHOURS 4 16 NU 1 E (HOURS'BILLED)
  RATESCHED 1 25 NU
WORKAREA
  PARMDATA 36
    RT1 4 1 NU 2 E
    RT2 4 NU 2 E
    RT3 4 NU 2 E
    RT4 4 NU 2 E
    RT5 4 NU 2 E
    RT6 4 NU 2 E
    RT7 4 NU 2 E
    RT8 4 NU 2 E
    RT9 4 NU 2 E
    AMOUNT 4 PD 2 E
    COMMENT 20 ( )
    BILLED AMT 5 PD 3 E
  CONTROL DEPTNO DIVNO
  ON ONE
    MOVE DYLPARM TO PARMDATA
  ENDONE
  IF RCODE EQ 'L'
    INX=(RATESCHED-1)*4
    BILLED AMT=PHOURS*RT1(INX) ROUNDED
    PICNSAVE REP3 USING PROECTNO EMPLNO PHOURS RATESCHED
    BILLED AMT
    BY 10
    REJECT
  ENDIF
  AMOUNT = HOURS*RATE ROUNDED
PICNSAVE REP2 USING DIVNO DEPTNO EMPLNO EMPLNAME HOURS
  SHIFT RATE AMOUNT
  BY 10
  LIST SUPPRESS (DIVNO DEPTNO) EMPLNO EMPLNAME HOURS SHIFT
    RATE AMOUNT
  ON CHANGE IN DEPTNO
    MOVE ' TOTAL DEPARTMENT' TO COMMENT

```

The REPORT statement indicates that the report is spooled to SYS280RS rather than SYS280R. The REPORT statement for a first request is not required unless overrides are needed.

All records whose RCODE is equal to L are passed to a subsequent request as REP3. Only fields specified in the PICNSAVE statement are passed. The REP3 data are blocked 10.

All records not equal to an RCODE of L are passed to a subsequent request as REP2. Only fields specified in the PICNSAVE statement are passed. The REP2 data are blocked 10.

Figure 414 z/OS Example 2 Using PICNSAVE (Page 1 of 3)


```

COMPUTER ASSOCIATES VISION:RESULTS 5.0                                DATE 1/15/01
1-----VISION:RESULTS FREE FORM TEXT-----72
LISTBRK:
  LIST COMMENT AT DEPTNO SUM (HOURS AMOUNT) WITH 1 BEFORE 3 AFTER
ON CHANGE IN DIVNO
  MOVE '** DIVISION TOTAL **' TO COMMENT
  GOTO LISTBRK
ON FINAL
  MOVE '*** GRAND TOTAL ***' TO COMMENT
  GOTO LISTBRK
T1 'PAYROLL DETAIL BY DIVISION / DEPARTMENT'
T2 'PERIOD ENDING - ,
T2+57 DYLDATE
NEWPAGE

```

The NEWPAGE command forces a page eject on the source listing output.

```

COMPUTER ASSOCIATES VISION:RESULTS 5.0                                DATE 1/15/01
1-----VISION:RESULTS FREE FORM TEXT-----72
REPORT2 80 WIDE
USE REP2;< USE STATEMENT
CONTROL EMPLNO
  SORT REP2 USING EMPLNO SHIFT
  LIST SUPPRESS EMPLNO (EMPLOYEE'NUMBER) SUPPRESS EMPLNAME
    DIVNO DEPTNO HOURS (HOURS'WORKED)
    SHIFT RATE (HOURLY'RATE) AMOUNT (GROSS'WAGES) ' '
ON CHANGE IN EMPLNO
  LIST '* EMPLOYEE TOTAL *' AT EMPLNO SUM HOURS SUM AMOUNT '***'
    WITH 2 AFTER
ON FINAL
  LIST '** FINAL TOTAL **' AT EMPLNO SUM HOURS SUM AMOUNT '***'
    WITH 3 BEFORE
T1 'P A Y R O L L   D E T A I L   B Y   E M P L O Y E E   '
T2 'PERIOD ENDING - '
T2+70 DYLDATE
NEWPAGE

```

Report request 2 is using the REP2 PICNSAVE file from the initial request as its input. All field definitions are passed and supported.

Figure 414 z/OS Example 2 Using PICNSAVE (Page 2 of 3)

```

COMPUTER ASSOCIATES VISION:RESULTS 5.0                                DATE 1/15/01
1-----VISION:RESULTS FREE  FORM  TEXT-----72
REPORT 80 WIDE SYS280RS
OPTION PRINTEROR
USE REP3 ; <- USE STATEMENT
WORKAREA
  PARMDATA 36
    RT1 4 1 NU 2 E
    RT2 4 NU 2 E
    RT3 4 NU 2 E
    RT4 4 NU 2 E
    RT5 4 NU 2 E
    RT6 4 NU 2 E
    RT7 4 NU 2 E
    RT8 4 NU 2 E
    RT9 4 NU 2 E
CONTROL PROJECTNO
ON ONE
  MOVE DYLPARM TO PARMDATA
ENDONE
SORT REP3 USING PROJECTNO EMPLNO
LIST SUPPRESS (PROJECTNO EMPLNO) PHOURS RATESCHED BILLED_AMT ' '
ON CHANGE IN PROJECTNO
  LIST ' ' AT PROJECTNO SUM (PHOURS BILLED_AMT) '***'
  WITH 1 BEFORE AND 3 AFTER
ON FINAL
  LIST ' ' AT PROJECTNO SUM (PHOURS BILLED_AMT) '****'
  WITH 2 BEFORE
T1 'LABOR COST BY PROJECT FOR PERIOD ENDING -'
  WITH 2 AFTER
T2 '      RATE SCHEDULES: #1          #2          #3' FIXED
T3 '                        #4          #5          #6' FIXED
T4 '                        #7          #8          #9' FIXED
T1+70 DYLDATE
T2+24 RT1
T2+35 RT2
T2+46 RT3
T3+24 RT4
T3+35 RT5
T3+46 RT6
T4+24 RT7
T4+35 RT8
T4+46 RT9

```

Report request 3 is using the REP3 PICNSAVE file from the initial request as its input. All field definitions are passed and supported.

Figure 414 z/OS Example 2 Using PICNSAVE (Page 3 of 3)

JCL for Example 2

```

//RESULTS JOB .....
//STEP01 EXEC PGM=DYL280
//STEPLIB DD DSN=your.DYL280.loadlib,DISP=SHR
//SYS280R DD SYSOUT=A
//SYS280RS DD SYSOUT=A,DCB=BLKSIZE=80
//SYSPRINT DD SYSOUT=A
//AUDPRINT DD SYSOUT=A
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//AUDWORK DD UNIT=SYSDA,SPACE=(TRK,(10,5))
//AUDEPF DD UNIT=SYSDA,SPACE=(TRK,(10,5)),
// DCB=(BLKSIZE=800,LRECL=80,RECFM=FB)
//AUDCBF DD UNIT=SYSDA,SPACE=(TRK,(10,5)),
// DCB=BLKSIZE=1000
//SYSOUT DD SYSOUT=A
//SORTLIB DD DSN=your.sortlib.dataset,DISP=SHR
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(5),,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(5),,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(5),,CONTIG)
//PAYLAB DD DSN=PAYROLL.LABEL,DISP=SHR
//REP2 DD UNIT=SYSDA,SPACE=(TRK,(10,5))
//REP3 DD UNIT=SYSDA,SPACE=(TRK,(10,5))
//SYSIN DD *

```

Example 2 VISION:Results program

Figure 415 z/OS JCL for [Example 2](#)

VSE

Example 1

```

COMPUTER ASSOCIATES VISION:RESULTS 5.0                                DATE 1/15/01
1-----VISION:RESULTS FREE  FORM  TEXT-----72
* VSE EXAMPLE...USING THE KEYWORD RETAIN AND THE COMMAND USE
OPTION DATA ' 620 670 754 818 850 960120015001800'
REPORT 80 WIDE
FILE PAYLAB FB 80 1600 DISK 3340 SYS012 RETAIN ;<- FILE STATEMENT
  DIVNO 2 (DIVN)
  DEPTNO 3 (DEPT)
  EMPLNO 5 (EMPL#)
  EMPLNAME 20 (NAME)
  HOURS 4 35 NU 1 E
  SHIFT 1 40 (SHF)
  RATE 4 45 NU 2 E
  RCODE 1 53
  PROJECTNO 5 11 (PROJECT'NUMBER)
  PHOURS 4 16 NU 1 E (HOURS'BILLED)
  RATESCHED 1 25 NU
WORKAREA
  PARMDATA 36
    RT1 4 1 NU 2 E
    RT2 4 NU 2 E
    RT3 4 NU 2 E
    RT4 4 NU 2 E
    RT5 4 NU 2 E
    RT6 4 NU 2 E
    RT7 4 NU 2 E
    RT8 4 NU 2 E
    RT9 4 NU 2 E
  AMOUNT 4 PD 2 E
  COMMENT 20 ()
CONTROL DEPTNO DIVNO
  IF RCODE EQ 'L' REJECT ENDIF
  AMOUNT = HOURS*RATE ROUNDED
  LIST SUPPRESS (DIVNO DEPTNO) EMPLNO EMPLNAME HOURS SHIFT
    RATE AMOUNT
ON CHANGE IN DEPTNO
  MOVE ' TOTAL DEPARTMENT' TO COMMENT
LISTBRK:
  LIST COMMENT AT DEPTNO SUM (HOURS AMOUNT) WITH 1 BEFORE 3 AFTER
ON CHANGE IN DIVNO
  MOVE '** DIVISION TOTAL **' TO COMMENT
  GOTO LISTBRK
ON FINAL
  MOVE '*** GRAND TOTAL ***' TO COMMENT
  GOTO LISTBRK
T1 'PAYROLL DETAIL BY DIVISION / DEPARTMENT'
T2 'PERIOD ENDING -'
T2+57 DYLDATE

```

The **REPORT** statement indicates the first report request in a multiple-report run (not required for the first request).

The keyword **RETAIN** passes the file attributes and field definitions to subsequent requests.

Figure 416 VSE Example 1 Using the FILE RETAIN (Page 1 of 3)

```

COMPUTER ASSOCIATES VISION:RESULTS 5.0                                DATE 1/15/01
1-----VISION:RESULTS FREE FORM TEXT-----72
REPORT2 80 WIDE
USE PAYLAB ; <-- USE STATEMENT
WORKAREA
  PARMDATA 36
    RT1 4 1 NU 2 E
    RT2 4 NU 2 E
    RT3 4 NU 2 E
    RT4 4 NU 2 E
    RT5 4 NU 2 E
    RT6 4 NU 2 E
    RT7 4 NU 2 E
    RT8 4 NU 2 E
    RT9 4 NU 2 E
  AMOUNT 4 PD 2 E
  COMMENT 20 ( )
CONTROL EMPLNO
  IF RCODE EQ 'L' REJECT ENDIF
  SORT PAYLAB USING EMPLNO SHIFT
  AMOUNT=HOURS*RATE ROUNDED
  LIST SUPPRESS EMPLNO (EMPLOYEE'NUMBER) SUPPRESS EMPLNAME
    DIVNO DEPTNO HOURS (HOURS'WORKED)
    SHIFT RATE (HOURLY'RATE) AMOUNT (GROSS'WAGES) ' '
ON CHANGE IN EMPLNO
  LIST SUM HOURS SUM AMOUNT '***' WITH 2 AFTER
ON FINAL
  LIST '*** FINAL TOTAL ***' AT EMPLNO SUM HOURS SUM AMOUNT '***'
    WITH 3 BEFORE
T1 'PAYROLL DETAIL EMPLOYEE'
T2 'PERIOD ENDING - '
T2+70 DYLDATE
REPORT3 80 WIDE
OPTION PRINTERERROR
USE PAYLAB ; <-- USE STATEMENT
WORKAREA
  PARMDATA 36
    RT1 4 1 NU 2 E
    RT2 4 NU 2 E
    RT3 4 NU 2 E
    RT4 4 NU 2 E
    RT5 4 NU 2 E
    RT6 4 NU 2 E
    RT7 4 NU 2 E
    RT8 4 NU 2 E
    RT9 4 NU 2 E
  BILLED AMT 5 PD 3 E
  COMMENT 20 ( )
CONTROL PROJECTNO
ON ONE
  MOVE DYLPARM TO PARMDATA
ENDONE

```

The REPORT2 statement is the beginning of the second request.

The USE statement indicates which file is to be processed in the request.

The REPORT3 statement is the beginning of the third request.

The USE statement indicates which file is to be processed in the request.

Figure 416 VSE Example 1 Using the FILE RETAIN (Page 2 of 3)

```

COMPUTER ASSOCIATES VISION:RESULTS 5.0                                DATE 1/15/01
1-----VISION:RESULTS FREE FORM TEXT-----72
      IF RCODE NE 'L' REJECT ENDIF
SORT PAYLAB USING PROJECTNO EMPLNO
INX (RATESCHED-1)*4
BILLED AMT=PHOURS*RT1(INX) ROUNDED
LIST SUPPRESS (PROJECTNO EMPLNO) PHOURS RATESCHED BILLED_AMT ' '
ON CHANGE IN PROJECTNO
      LIST SUM (PHOURS BILLED_AMT) '***'
      WITH 1 BEFORE AND 3 AFTER
ON FINAL
      LIST SUM (PHOURS BILLED_AMT) '****'
      WITH 2 BEFORE
T1 'LABOR COST BY PROJECT FOR PERIOD ENDING - '
      WITH 2 AFTER
T2 '      RATE SCHEDULES: #1          #2          #3' FIXED
T3 '                        #4          #5          #6' FIXED
T4 '                        #7          #8          #9' FIXED
T1+70 DYLDATE
T2+24 RT1
T2+35 RT2
T2+46 RT3
T3+24 RT4
T3+35 RT5
T3+46 RT6
T4+24 RT7
T4+35 RT8
T4+46 RT9

```

Figure 416 VSE Example 1 Using the FILE RETAIN (Page 3 of 3)

PAYROLL DETAIL BY DIVISION / DEPARTMENT							01/15/01	
DIVN	DEPT	EMPL#	NAME	PERIOD ENDING -	HOURS	SHF	RATE	AMOUNT
01	127	01574	LORAC, M.B.		37.5	3	9.75	365.63
		03200	SQUIGLEY, GEO.D.		40.0	1	11.50	460.00
		23941	POOLE, M.T.		40.0	1	7.50	300.00
		23941	POOLE, M.T.		10.5	2	11.25	118.13
		TOTAL DEPARTMENT			128.0			1,243.76
01	432	00009	MUMFORD, M.E.		40.0	1	8.50	330.00
		00009	MUMFORD, M.E.		9.6	3	12.75	122.40
		12370	PEREZ, JUAN PERICO		37.5	1	7.50	281.25
		62134	GODFREY, Q.Z.		40.0	2	7.50	300.00
		65656	YELNATS, STANLEY		40.0	2	9.25	370.00
		TOTAL DEPARTMENT			167.1			1,403.65
		** DIVISION TOTAL **			295.1			2,647.41
02	951	07123	ZAK, HERMAN P.		40.0	2	8.00	320.00
		07123	ZAK, HERMAN P.		6.0	3	12.00	72.00
		32785	JONES, ROBERT S.		39.0	1	10.25	399.75
		74424	REEVES, SLADE		40.0	1	9.00	360.00
		91101	WONG, LING SU		40.0	1	7.50	300.00
		97537	VON SCHMIDT, JOHANN		40.0	1	7.50	300.00
		TOTAL DEPARTMENT			205.0			1,751.75
02	952	04010	WASHINGTON, LEROY		32.0	2	9.75	312.00
		04012	SVENSEN, ERICA T.		40.0	2	9.50	380.00
		77778	STIVIC, G.N.		40.0	2	8.50	340.00
		TOTAL DEPARTMENT			112.0			1,032.00
		** DIVISION TOTAL **			317.0			2,783.75
		*** GRAND TOTAL ***			612.1			5,431.16

Figure 417 VSE Example 1 Output from VSE (Page 1 of 3)

P A Y R O L L		D E T A I L		P E R I O D		B Y E M P L O Y E E		E N D I N G -		S H F		H O U R L Y		G R O S S		01/15/01	
EMPLOYEE NUMBER	NAME	DIVN	DEPT	HOURS WORKED	SHF	HOURLY RATE	GROSS WAGES										
00009	MUMFORD, M.E.	01	432	40.0	1	8.50	330.00										
		01	432	9.6	3	12.75	122.40										
				49.6			452.40										
01574	LORAC, M.B.	01	127	37.5	3	9.75	365.63										
				37.5			365.63										
03200	SQUIGLEY, GEO.D.	01	127	40.0	1	11.50	460.00										
				40.0			460.00										
04010	WASHINGTON, LEROY	02	952	32.0	2	9.75	312.00										
				32.0			312.00										
04012	SVENSEN, ERICA T.	02	952	40.0	2	9.50	380.00										
				40.0			380.00										
07123	ZAK, HERMAN P.	02	951	40.0	2	8.00	320.00										
		02	951	6.0	3	12.00	72.00										
				46.0			392.00										
12370	PEREZ, JUAN PERICO	01	432	37.5	1	7.50	281.25										
				37.5			281.25										
23941	POOLE, M.T.	01	127	40.0	1	7.50	300.00										
		01	127	10.5	2	11.25	118.13										
				50.5			418.13										
32785	JONES, ROBERT S.	02	951	39.0	1	10.25	399.75										
				39.0			399.75										
62134	GODFREY, Q.Z.	01	432	40.0	2	7.50	300.00										
				40.0			300.00										
65656	YELNATS, STANLEY	01	432	40.0	2	9.25	370.00										
				40.0			370.00										
74424	REEVES, SLADE	02	951	40.0	1	9.00	360.00										
				40.0			360.00										
77778	STIVIC, G.N.	02	952	40.0	2	8.50	340.00										
				40.0			340.00										
91101	WONG, LING SU	02	951	40.0	1	7.50	300.00										
				40.0			300.00										
97537	VON SCHMIDT, JOHANN	02	951	40.0	1	7.50	300.00										
				40.0			300.00										
** FINAL TOTAL **				612.1			5,431.16										

Figure 417 VSE Example 1 Output from VSE (Page 2 of 3)

LABOR COST BY PROJECT FOR PERIOD ENDING - 01/15/01

RATE SCHEDULES:		#1 7.50	#2 8.00	#3 8.50	
		#4 9.00	#5 9.25	#6 9.50	
		#7 9.75	#8 10.25	#9 11.50	
PROJECT NUMBER	EMPL#	HOURS BILLED	RATESCHED	BILLED_AMT	
19	01574	20.0	7	195.000	
	04010	28.9	7	281.780	
	07123	6.0	2	48.000	
	12370	30.0	1	225.000	
	23941	10.5	1	78.750	
	65656	12.6	5	116.550	
	91101	40.0	1	300.000	
		148.0		1,245.080	**
110	03200	40.0	9	460.000	
	04010	3.1	7	30.225	
	07123	32.0	2	256.000	
	23941	13.0	1	97.500	
	32785	20.0	8	205.000	
		108.1		1,048.725	**
162	00009	20.0	3	170.000	
	01574	17.5	7	170.625	
	65656	10.0	5	92.500	
		47.5		433.125	**
170	00009	20.0	3	170.000	
	65656	5.4	5	49.950	
	77778	17.0	3	144.500	
	97537	27.6	1	207.000	
		70.0		571.450	**
409	00009	9.6	3	81.600	
	12370	7.5	1	56.250	
	32785	19.0	8	194.750	
	65656	12.0	5	111.000	
	77778	19.2	3	163.200	
	97537	8.2	1	61.500	
		75.5		668.300	**

Figure 417 VSE Example 1 Output from VSE (Page 3 of 3)

```
// JOB RESULTS
// ASSGN SYS012,X'cuu'
// DLBL PAYLAB,'PAYROLL.LABEL'
// EXTENT SYS012
// DLBL IJSYS04,'VISION:RESULTS WORK FILE',0
// EXTENT SYS004,...extent information
// ASSGN SYS008,X'cuu'
// DLBL SYS008,'VISION:RESULTS WORK FILE2',0
// EXTENT SYS008,...extent information (20 tracks)
// ASSGN SYS009,X'cuu'
// DLBL SYS009,'VISION:RESULTS LETTERS.SYSLST',0
// EXTENT SYS009,...extent information (100 tracks)
// ASSGN SYS010,X'cuu'
// DLBL SYS010,'VISION:RESULTS WORK FILE3',0
// EXTENT SYS010,...extent information (20 tracks)
// ASSGN SYS011,X'cuu'
// DLBL SYS011,'VISION:RESULTS WORK FILE4',0
// EXTENT SYS011,...extent information (20 tracks)
// DLBL SORTWK1,'VISION:RESULTS SORT WORK',0
// EXTENT SYS001,...extent information
// EXEC DYL280,SIZE=250K
// Example 1 VISION:Results program
/*
/ &
```

Figure 418 VSE JCL for [Example 1](#)

Example 2

```

COMPUTER ASSOCIATES VISION:RESULTS 5.0                                DATE 1/15/01
1-----VISION:RESULTS FREE FORM TEXT-----72
* VSE EXAMPLE...USING PICNSAVE AND USE COMMANDS
OPTION DATA ' 620 670 754 818 850 960120015001800'
REPORT 80 WIDE
FILE PAYLAB FB 80 1600 DISK 3340 SYS012 ; <- FILE STATEMENT
DIVNO 2 (DIVN)
DEPTNO 3 (DEPT)
EMPLNO 5 (EMPL#)
EMPLNAME 20 (NAME)
HOURS 4 35 NU 1 E
SHIFT 1 40 (SHF)
RATE 4 45 NU 2 E
RCODE 1 53
PROJECTNO 5 11 (PROJECT'NUMBER)
PHOURS 4 16 NU 1 E (HOURS'BILLED)
RATESCHED 1 25 NU
WORKAREA
PARMDATA 36
RT1 4 1 NU 2 E
RT2 4 NU 2 E
RT3 4 NU 2 E
RT4 4 NU 2 E
RT5 4 NU 2 E
RT6 4 NU 2 E
RT7 4 NU 2 E
RT8 4 NU 2 E
RT9 4 NU 2 E
AMOUNT 4 PD 2 E
COMMENT 20 ( )
BILLED AMT 5 PD 3 E
CONTROL DEPTNO DIVNO
ON ONE
MOVE DYLPARM TO PARMDATA
ENDONE
IF RCODE EQ 'L'
INX=(RATESCHED-1)*4
BILLED AMT=PHOURS*RT1(INX) ROUNDED
PICNSAVE REP3 USING PROJECTNO EMPLNO PHOURS RATESCHED
BILLED AMT
BY 10 SYS015 DISK 3330
REJECT
ENDIF
AMOUNT = HOURS*RATE ROUNDED
PICNSAVE REP 2 USING DIVNO DEPTNO EMPLNO EMPLNAME HOURS
SHIFT RATE AMOUNT
BY 10 SYS014 DISK 3330
LIST SUPPRESS (DIVNO DEPTNO) EMPLNO EMPLNAME HOURS SHIFT
RATE AMOUNT
COMPUTER ASSOCIATES VISION:RESULTS 5.0                                DATE 1/15/01
1-----VISION:RESULTS FREE FORM TEXT-----72
ON CHANGE IN DEPTNO
MOVE ' TOTAL DEPARTMENT' TO COMMENT
LISTBRK:
LIST COMMENT AT DEPTNO SUM (HOURS AMOUNT) WITH 1 BEFORE 3 AFTER
ON CHANGE IN DIVNO
MOVE '** DIVISION TOTAL **' TO COMMENT
GOTO LISTBRK
ON FINAL
MOVE '*** GRAND TOTAL ***' TO COMMENT
GOTO LISTBRK
T1 'PAYROLL DETAIL BY DIVISION / DEPARTMENT'
T2 'PERIOD ENDING -'
T2+57 DYLDATE
NEWPAGE

COMPUTER ASSOCIATES VISION:RESULTS 5.0                                DATE 1/15/01
1-----VISION:RESULTS FREE FORM TEXT-----72
REPORT 80 WIDE
USE REP2 ; <--USE STATEMENT
CONTROL EMPLNO
SORT REP2 USING EMPLNO SHIFT
LIST SUPPRESS EMPLNO (EMPLOYEE'NUMBER) SUPPRESS EMPLNAME

```

The **REPORT** statement indicates the first report request in a multiple report run (not required for the first request).

All records whose RCODE is equal to L are passed to a subsequent request as REP3. Only fields specified in the PICNSAVE statement are passed. The REP3 data are blocked 10.

All records not equal to an RCODE of L are passed to a subsequent request as REP2. Only fields specified in the PICNSAVE statement are passed. The REP2 data are blocked 10.

The **NEWPAGE** command forces a page eject on the source listing output.

Report request 2 is using the REP2 PICNSAVE file from the initial request as its input. All field definitions are passed and supported.

Figure 419 VSE Example 2 Using PICNSAVE (Page 1 of 2)


```

          DIVNO DEPTNO HOURS (HOURS'WORKED)
          SHIFT RATE (HOURLY'RATE) AMOUNT (GROSS'WAGES) ' '
ON CHANGE IN EMPLNO
  LIST '* EMPLOYEE TOTAL *' AT EMPLNO SUM HOURS SUM AMOUNT '***'
    WITH 2 AFTER
ON FINAL
  LIST '** FINAL TOTAL **' AT EMPLNO SUM HOURS SUM AMOUNT '***'
    WITH 3 BEFORE
T1 'P A Y R O L L D E T A I L B Y E M P L O Y E E '
T2 'PERIOD ENDING - '
T2+70 DYLDATE
NEWPAGE
COMPUTER ASSOCIATES VISION:RESULTS 5.0 DATE 1/15/01
1-----VISION:RESULTS FREE FORM TEXT-----72
REPORT3 80 WIDE
OPTION PRINTERORR
USE REP3 ; <- USE STATEMENT
WORKAREA
  PARMDATA 36
    RT1 4 1 NU 2 E
    RT2 4 NU 2 E
    RT3 4 NU 2 E
    RT4 4 NU 2 E
    RT5 4 NU 2 E
    RT6 4 NU 2 E
    RT7 4 NU 2 E
    RT8 4 NU 2 E
    RT9 4 NU 2 E
CONTROL PROJECTNO
ON ONE
  MOVE DYLPARM TO PARMDATA
ENDONE
SORT REP3 USING PROJECTNO EMPLNO
LIST SUPPRESS (PROJECTNO EMPLNO) PHOURS RATESCHED BILLED_AMT ' '
ON CHANGE IN PROJECTNO
  LIST ' ' AT PROJECTNO SUM (PHOURS BILLED_AMT) '***'
    WITH 1 BEFORE AND 3 AFTER
ON FINAL
  LIST ' ' AT PROJECTNO SUM (PHOURS BILLED_AMT) '****'
    WITH 2 BEFORE
T1 'LABOR COST BY PROJECT FOR PERIOD ENDING - '
  WITH 2 AFTER
T2 '    RATE SCHEDULES: #1          #2          #3' FIXED
T3 '                      #4          #5          #6' FIXED
T4 '                      #7          #8          #9' FIXED
T1+70 DYLDATE
T2+24 RT1
T2+35 RT2
T2+46 RT3
T3+24 RT4
T3+35 RT5
T3+46 RT6
T4+24 RT7
T4+35 RT8
T4+46 RT9

```

Report request 3 is using the REP3 PICNSAVE file from the initial request as its input. All field definitions are passed and supported.

Figure 419 VSE Example 2 Using PICNSAVE (Page 2 of 2)

```
// JOB RESULTS
// ASSGN SYS012,X'cuu'
// DLBL PAYLAB,'PAYROLL.LABEL'
// EXTENT SYS012
// DLBL IJSYS04,'VISION:RESULTS WORK FILE',0
// EXTENT SYS004,...extent information
// ASSGN SYS008,X'cuu'
// DLBL SYS008,'VISION:RESULTS WORK FILE2',0
// EXTENT SYS008,...extent information (20 tracks)
// ASSGN SYS009,X'cuu'
// DLBL SYS009,'VISION:RESULTS LETTERS.SYSLST',0
// EXTENT SYS009,...extent information (100 tracks)
// ASSGN SYS010,X'cuu'
// DLBL SYS010,'VISION:RESULTS WORK FILE3',0
// EXTENT SYS010,...extent information (20 tracks)
// ASSGN SYS011,X'cuu'
// DLBL SYS011,'VISION:RESULTS WORK FILE4',0
// EXTENT SYS011,...extent information (20 tracks)
// DLBL SORTWK1,'VISION:RESULTS SORT WORK',0
// EXTENT SYS001,...extent information
// ASSGN SYS014,X'cuu'
// DLBL REP2,'VISION:RESULTS TEMPORARY SAVE1',0
// EXTENT SYS014,...extent information
// ASSGN SYS015,X'cuu'
// DLBL REP3,'VISION:RESULTS TEMPORARY SAVE2',0
// EXTENT SYS015,...extent information
// EXEC DYL280,SIZE=250K
      Example 2 VISION:RESULTS program
/*
/&
```

Figure 420 VSE JCL for [Example 2](#)

Chapter 19: File Print Commands

Use the file and field print feature to:

- Print data in graphics, hex and graphics, or hex only representations.
- Print entire records, portions of records, or specific fields in records.
- Print messages from your program.
- Print only selected records, every nth record, or the first nnn records.

The records or fields being printed can be from files of fixed variable, variable spanned (z/OS), or undefined format. The files can be of sequential, BDAM, [ISAM](#), or VSAM (KSDS, ESDS, RRDS) organization. Literals or fields in work areas can be printed also.

Example 1 Printing in Graphics

```
COMPUTER ASSOCIATES  *VISION:RESULTS 5.0*          DATE 1/15/01    PAGE 1
 1          2          3          4          5          6          7          8          9
1234567890123456789012345678901234567890123456789012345678901234567890
PRINT 07C8006547TORRES      ERNESTO  E021580042480000000 00000000000 0007870000    TORRES,ERNESTO
1
      23444 PARK LANE      LOS ANGELES CA      %      BOBO 000000
          PARASO, B MD          A
```

Figure 421 Graphics Printing Example

Example 2 Printing in Hex and Graphics

[illegible]

Figure 422 Hex and Graphics Printing Example

Example 3 Printing in Hex Only

[illegible]

Figure 423 Hex Printing Example

The file print feature is independent of report writing. You can do both in the same run. The print is directed to the SYSPRINT file for z/OS and SYSLST for VSE. If you are using the VSE environment and creating a report in your program along with file printing, you can find your report and file print interleaved, unless your report is assigned elsewhere. See [Chapter 16: Using Report Statements](#).

There are two mutually exclusive ways to print data in your program:

- Using the `OPTION` statement: `OPTION PRINT FILEIN`. This method works fine if you are printing records from one file and have no other kind of printing to do.
- Using `print` immediate commands within the program logic. This method is advisable when your printing is conditional or when you have different types of data to print. For example, in one program you want to print certain records, error messages, and specific fields at various times. Or, you can vary the type of print by printing records in hexadecimal and graphics representation and messages in graphics only. The `PRINT` commands used in your logic allow for this.

Printing Using the Option Statement

Only entire records from a file can be printed using the OPTION statement. For additional information, see [Chapter 4: Using the OPTION Statement](#). The formats are as follows:

```
OPTION PRINT [filename]
```

Figure 424 PRINT Graphics Format

Prints an entire accepted record in graphics format. If you omit the file name, records from the first file defined by a FILE statement are printed.

```
OPTION HEXPRINT [filename]
```

Figure 425 PRINT Hex and Graphics Format

Prints an entire accepted record in hex and graphics format. If you omit the file name, records from the first file defined by a FILE statement are printed.

```
OPTION HEX [filename]
```

Figure 426 PRINT Hex only Format

Prints an entire accepted record in hex only format. If you omit the file name, records from the first file defined by a FILE statement are printed.

```
OPTION LCPRINT [filename]
```

Figure 427 PRINT Graphics Format with Lowercase Characters

Prints an entire accepted record in graphics format. If the data contains lowercase characters, they print as well. If you are printing using OPTION PRINT, lowercase and other often unprintable characters are translated out to speed up printing.

```
OPTION REPORTFILE [filename]
```

Figure 428 PRINT Format for VSE

VSE only. Prints an entire accepted record, treating the first byte of each record as containing ASA or machine carriage control. This option is useful for printing report files that have been previously written to tape or disk.

Examples of printing using the OPTION statement:

```
OPTION PRINT                                VSE
FILE FILEIN FB 250 2500 SYS012;
```

Above, a file print utility is being specified. Every record from the input file is printed in graphics format.

```
OPTION HEXPRINT VSAMIN
FILE VSAMIN KSDS F 210
```

Above, a VSAM KSDS file is printed in its entirety.

```
OPTION HEXPRINT SEQIN
FILE SEQIN FB 100 1000 SYS014;
ACCOUNTNUM 5 7 NU
IF ACCOUNTNUM NE 4001 REJECT ENDIF
```

VSE

Above, an input file is being read. However, only records with the Account Number 4001 are being accepted. The 4001 records are the ones that are printed in hexadecimal and graphics.

```
OPTION HEXPRINT FILEOUT
FILE FILEIN FB 72 720
NAME 20 ADDRESS 30 CITY 15 STATE 2 ZIP 5
FILE FILEOUT OUTPUT FB 92 920 FROM FILEOUT SYS021
NAMEO 30 ADDRESSO 40 CITYO 15 STATE 2 ZIP 5

MOVE NAME TO NAMEO
MOVE ADDRESS TO ADDRESSO
MOVE CITY LENGTH 22 TO CITYO
```

Above, an input record is being reformatted and written to FILEOUT. In addition, during the VISION:Results output cycle, before the record is written, it is printed in hex and graphics format.

Printing Using the Print Immediate Commands

By issuing various print commands in your program, you can cause immediate printing of records, fields, or messages. You can print indexed fields, as well as multiple objects (such as file names, fields, and literals) in any combination. The format for these PRINT immediate commands is as follows:

```
{PRINT | HEXPRINT | HEX | LCPRINT | REPORTFILE}

{filename1 | dataname1 | [(index)] | 'literal1'}
[LENGTH {dataname | nnn}]

{filename2 | dataname2 | [(index)] | 'literal2'}
[LENGTH {dataname | nnn}]

{filename3 | dataname3 | [(index)] | 'literal3'}
[LENGTH {dataname | nnn}]
```

Figure 429 PRINT Immediate Commands

PRINT	Prints the data immediately in graphics only representation.
HEXPRINT	Prints the data immediately in hexadecimal and graphics representation.
HEX	Prints the data immediately in hexadecimal representation.
LCPRINT	Prints the data immediately in graphics representation and prints lowercase and other normally non-printable characters.
REPORTFILE	VSE only. Prints the record and treats the first character of the record as a carriage control character. This print option is useful for printing report files where the first byte of each record contains machine or ASA carriage control.

Example 4 Print Command

```

OPTION

FILE FILE1 FB 80
    RECORD1 80

FILE FILE2 FB 80
    RECORD2 80

FILE FILE3 FB 80
    DATA3A      10
    FILL1        24
    DATA3B      3  PD
    FILL2        20
    DATA3C      4  BI
    DATA3D      3  PD
    FILL3        16

WORKAREA
    DATA4A      20
    DATA4B      20
    DATA4C      20
    DATA4D      20

*
* PRINT IMMEDIATE STUFF
*

PRINT    FILE1    FILE2

HEX      RECORD1 RECORD2

HEXPRINT DATA3A  DATA3B  DATA3C  DATA3D

INX = 0

STARTIT:
IF INX LE 60
    LCPRINT DATA4A(INX)
    HEXPRINT DATA4A(INX)
    INX = INX + 20
    GOTO STARTIT
ENDIF

FIN

```

Figure 430 Print Command Example

File Name, Data Name, Indexed Data Name, or Literal

An entry is required. Following the PRINT command, specify a file name, data name, indexed data name, or literal to identify what is to be printed. More than one file name, data name or literal can be included in the PRINT command, but no separation exists between objects on the print line.

If you code a file name, the current record from the specified file is printed. If you code a data name, the field identified by this name is printed. A numeric, packed, or binary field is not converted to printable data automatically prior to its being printed, as is the case if the field is LISTed in a report. You also can code a 1- to 70-character literal enclosed in quotation marks. You can use a literal to separate print objects on the output print line.

Examples:

```

HEXPRINT ISFILE
PRINT ACCTNUM
PRINT 'ERROR 501'

```

LENGTH n or Data Name

The LENGTH keyword, followed by a numeric value or data name, identifies a field containing the length of the field or record you want to print. This can be coded to override the length that VISION:Results normally prints. If you code a data name, define this field in a record or work area. If it is the same data name you coded in a LENGTH dataname keyword in a FILE statement where VISION:Results automatically allocates the field, you do not have to define it.

The maximum length that can be printed is 32767 (32752 for variable-length records).

Examples:

```
HEXPRINT ISFILE LENGTH 30
```

Only the first 30 bytes of a file are printed.

```
PRINT ACCTNUM LENGTH 17
```

Seventeen bytes of data, starting with the field called ACCTNUM, are printed in graphics format.

Examples of printing using PRINT commands:

```
FILE FILEIN FB 250 2500 SYS014  
PRINT FILEIN
```

Here, a file print utility is being specified. Every record from the input file is printed in graphics format.

```
FILE VSAMIN KSDS F 210  
HEXPRINT VSAMIN
```

A VSAM KSDS file is printed in its entirety in hexadecimal and graphics format.

```
FILE SYSIN CARDS STATUS CARDSTAT  
*CARD FIELD DEFINITIONS  
CARDKEY 5 1 NU  
FILE ISAMMST KSDS RANDOM FINDKEY F 150  
KEYLEN 5 KEYLOC 2 STATUS ISSTAT  
  
IF CARDSTAT EQ 'E' STOP ENDIF  
MOVE CARDKEY TO FINDKEY  
READ ISAMMST  
IF ISSTAT NE 'Y' GOTO ERRPRINT ENDIF  
HEXPRINT ISAMMST  
ACCEPTERRPRINT:  
PRINT 'NO RECORD FOUND'  
HEXPRINT FINDKEY  
ACCEPT
```


In z/OS, a VSAM KSDS file is being read randomly using keys that VISION:Results is reading in as instream data. If the record is found (ISSTAT EQ Y), it is printed in hexadecimal and graphics and another cycle is entered using ACCEPT. If the record is not found, an error routine first prints out NO RECORD FOUND on one line and then the bad key on the next line in hexadecimal and graphics. Then the next cycle is entered using the ACCEPT command.

```
ERRMSG 16 VALUE 'NO RECORD FOUND'
ERRACCT 5 NU
.
.
MOVE FINDKEY TO ERRACCT
PRINT ERRMSG LENGTH 21
.
.
```

An error message consisting of a constant and the erroneous account number is formatted and printed in graphics representation. The length printed includes both fields.

```
FILE VARIN INPUT VB 80 844 SYS011 LENGTH VARLEN;          VSE
RECFLD 1

HEXPRINT RECFLD LENGTH VARLEN
```

A variable-length file is printed in hexadecimal and graphics. This is the recommended method of printing when you want only the data portion of the record printed and not the Record Descriptor Word (RDW) prefix. To do this, use the LENGTH dataname keyword in the FILE statement, and in the PRINT command, specify a data name that identifies the first data byte (RECFLD) of the record. Following this, code the LENGTH phrase using the same data name you coded in the FILE statement. This approach is not necessary for VSAM variable-length records, as there is no RDW.

```
FILE VARIN VB 80 844 LENGTH LEN1
VARINREC 80
FILE VAROUT VB 80 844 OUTPUT FROM VAROUT LENGTH LEN2
VAROUTREC 80
MOVE VARINREC TO VAROUTREC LENGTH LEN1
MOVE LEN1 TO LEN2
HEXPRINT VAROUT
```

A variable-length output record is being printed in hexadecimal and graphics prior to its being written. To do this, it is important that the length be set in the LEN2 field before issuing the PRINT command.

```
FILE VARIN VB 80 844 SYS011
HEXPRINT VARIN
```

A variable-length file is being printed in hexadecimal and graphics. Both the Record Descriptor Word (RDW) and the data portion of the record are printed.

Other Print Features

Comments

Along with your data, you also can include up to 10 characters of comments to print to the left of the data. DYLPRTCOMM is a special 10-byte field of character data to which you can move whatever information you want before printing.

```
MOVE 'TRANS REC' TO DYLPRTCOMM
PRINT TRANSIN
MOVE 'MAST REC' TO DYLPRTCOMM
HEXPRINT MASTIN
```

Records from two files are printed. To better identify which is which, comments are included in the print line.

This feature applies to both printing using the OPTION statement and using print immediate commands.

If you do not move anything to the comments field before issuing the print command, VISION:Results prints the data name or file name in the comments portion of the line. However, if there is more than one data name in your print command, you must move the data names to DYLPRTCOMM before issuing your print command to have them appear in the comments area. This feature makes it easy to set up a useful trace during debugging. If you are using the OPTION print, comments are not filled in.

If you do not want the data name or file name to be printed, move an unprintable character to DYLPRTCOMM, for example, X'00.'

Record Number

Whenever you are printing a file either with OPTION or using print immediate commands, a record number is printed along with the record.

If you are printing an output record, the record number is one less than the actual number if the print occurs before the write. This is because the number of an output record is set after the write. Also, if you are printing records in a program that is being sorted, the record number can be meaningless, depending on when the print occurs.

Chapter 20: VSAM Processing

VISION:Results supports VSAM KSDS (key sequence data set), ESDS (entry sequence data set), and RRDS (relative record data set) files. Under the virtual storage access method (VSAM), record types can be fixed, variable, or variable spanned.

The input and/or output modes supported are sequential retrieval, sequential load/load extend/insert, random retrieval, random retrieval and update, limited sequential (skip) retrieval by full or generic key, as well as sequential retrieval with update.

- If you have specified the keyword VSAMCAT=Y in the DYLINSTL macro or OPTION VSAMCAT in your program, you must specify VSAM in the FILE statement.
- If you are using the VSAMCAT feature and the JCL contains the DD or DLBL statement for the VSAM file during a FREEZE, verify, or compile and go run, then you do not need to specify the file type (for example, KSDS, ESDS, or RRDS) and other characteristics that are usually required on the FILE statement. VISION:Results does this for you.
- If you are using the VSAMCAT feature and the JCL does not contain the DD or DLBL statement for the VSAM file during a FREEZE, verify, or compile and go run, then you need to specify the file type (for example, KSDS, ESDS, or RRDS) and other characteristics that must be specified on the FILE statement.
- If you are using dynamically allocated VSAM files (files that use the ALLOCATE command), then you need to specify the file type and other characteristics on the FILE statement.

For more information about the DYLINSTL macro, see the *Advantage VISION:Results for z/OS Reference Summary*. For information about the OPTION statement, see [Chapter 4: Using the OPTION Statement](#).

For z/OS, VSAM files using record-level sharing (RLS) require VSAMCAT.

Syntax Format

The keywords and operands pertaining only to VSAM ESDS, KSDS, or RRDS are described in this chapter. Operands such as filename, recordsize, or dataname are described in [Chapter 5: Data Name Qualification](#).

The examples provided show syntax for programming both with and without the OPTION VSAMCAT statement in your program or the keyword VSAMCAT=Y included in the DYLINSTL macro.

VSAM ESDS FILES

Throughout this section, items shown in **bold** are optional when using the OPTION VSAMCAT statement or the VSAMCAT=Y keyword in the DYLINSTL macro. Underlined items indicate the default value. When using the VSAMCAT feature, you must enter VSAM in the FILE statement. Otherwise, the VSAMCAT feature is not used for that file.

Note: In this example, the RANDOMX, SKIPX, and RELBYTEX keywords can be used in z/OS only.

```
FILE filename [VSAM]
ESDS [PASSWORD [{ 'password' | dataname2 }]]
{INPUT | OUTPUT FROM filename | IO}
[{F | V | S} recordsize]
[RANDOM dataname2 {CH | NU | BI | PD} [decimals] | RANDOMX dataname2 |
  SKIP dataname2 {CH | NU | BI | PD} [decimals] | SKIPX dataname2 SEQUENTIAL]
[RELBYTE dataname2] | RELBYTEX dataname2 [POSITION dataname2]
[STATUS dataname2] [COUNT dataname2] [LENGTH dataname2]
[DROPERR] [DUMMY] [REUSE | NULL]
[RETAIN | NORETAIN]
[{MODIFY modulename [nK]}] [SIZE nnnnn]
  [PARM ({IOAREA | WORK | dataname3...})] [DYNAM]
[SYSnnn]
```

Figure 431 VSAM ESDS Files

VSAM KSDS FILES

```
FILE filename [VSAM]
KSDS [PASSWORD { 'password' | dataname2 }]]
{INPUT | OUTPUT FROM filename | IO}
[{F | V | S} recordsize]
[RANDOM dataname2 {CH | NU | BI | PD} [decimals] |
  SKIP dataname2 {CH | NU | BI | PD} [decimals] | SEQUENTIAL]
[RELBYTE dataname2] | RELBYTEX dataname2 [POSITION dataname2]
[STATUS dataname2] [COUNT dataname2] [LENGTH dataname2]
[DROPERR] [DUMMY] [REUSE | NULL]
[RETAIN | NORETAIN]
[{MODIFY modulename [nK]}] [SIZE nnnnn]
  [PARM ({IOAREA | WORK | dataname3...})] [DYNAM]
[SYSnnn]
```

Figure 432 VSAM KSDS Files (Page 1 of 2)

```

[KEYLEN nnn [KEYLOC nnnn]] [POSITION dataname2]
[REUSE | NULL] [ERASE dataname2] [PARTKEY nnn]
[STATUS dataname2] [COUNT dataname2] [LENGTH dataname2]
[DROPERR] [DUMMY]
[RETAIN | NORETAIN]
[{MODIFY modulename [nK]} [SIZE nnnnn]
  [PARM ({IOAREA | WORK | dataname3...})]]] [DYNAM]
[SYSnnn]

```

Figure 432 VSAM KSDS Files (Page 2 of 2)

VSAM RRDS FILES

```

FILE filename [VSAM]
RRDS [PASSWORD {'password' | dataname2}]
{INPUT | OUTPUT FROM filename | IO}
[{F} [recordsize]]
[RANDOM dataname2 {CH | NU | BI | PD} [decimals] |
  SKIP dataname2 {CH | NU | BI | PD} [decimals] | SEQUENTIAL]
[RELREC dataname2] [POSITION dataname2] [ERASE dataname2]
[STATUS dataname2] [COUNT dataname2] [LENGTH dataname2]
[DROPERR] [DUMMY] [REUSE | NULL]
[RETAIN | NORETAIN]
[{MODIFY modulename [nK]} [SIZE nnnnn]
  [PARM ({IOAREA | WORK | dataname3...})]]] [DYNAM]
[SYSnnn]

```

Figure 433 VSAM RRDS Files

Creating a File

This section describes:

- The creation of a new VSAM file.
- The replacement of an old VSAM file.

If you intend to read or update an existing file only, you do not need to be concerned with this procedure.

To create either type of file, describe the file to the operating system using JCL (Job Control Language). For example:

```
(VSE) // ASSGN SYSnnn,X'cuu'  
      // DLBL MYFILE,'MY.FILE.IDENT', ...  
      // EXTENT SYSnnn, ...  
  
(z/OS) //MYFILE DD DSN=MY.FILE.IDENT,DISP= ...
```

Figure 434 VSAM JCL

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

In addition, for VSAM files, you must first execute a program supplied by IBM called IDCAMS (or Access Method Services) to DEFINE the attributes of the file and store that information in the VSAM Catalog. For instructions about using IDCAMS, see the IBM publication *Access Method Services User's Guide*. For example:

```
DEFINE CLUSTER (NAME (MY.FILE.IDENT) -  
  FILE (MYFILE) -  
  TRACKS (4,1) -  
  KEYS (5,1) -  
  RECSZ (40,40) -  
  REUSE -  
  SPEED) -  
  DATA (NAME (MY.FILE.DATA) -  
    FREESPACE (50,25)) -  
  INDEX (NAME (MY.FILE.INDEX))
```

Figure 435 IDCAMS (Access Method Services)

These statements define a key-sequenced (KSDS) VSAM file having a file name or ddname of MYFILE, and having fixed-length records 40 bytes long, a 5-byte key starting in location 2, and the ability to be reused (written over).

For the statements used to create VSAM files, see [Key Sequence VSAM \(KSDS\)](#). [Skip Sequential Read on page 493](#).

Key Sequence VSAM (KSDS) Create

Use VISION:Results statements to define the input file and the file to be created (output). If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code one of the following:

F	Fixed-length records
V	Variable-length records
S	Spanned variable-length records

VSAM records are never blocked.

```
FILE TAPEIN FB 40 4800  
FILE MYFILE KSDS OUTPUT FROM TAPEIN F REUSE
```

Figure 436 Key Sequence VSAM (KSDS)

The example above reads each record on the input file and copies it to the output file, unchanged. For information about selecting records, reformatting, or defining fields (data items), see *Advantage VISION:Results for z/OS Getting Started*. For more information about record selection and data manipulation, see [Chapter 7: Using the IF Command](#) and [Chapter 10: Using the MOVE Command](#).

The keyword REUSE causes any data already in the file (should such be the case) to be replaced by the new data. If REUSE is not specified, the new input records are inserted among the existing records according to the key. The key of a new record must not match that of an existing one. If REUSE is coded in your FILE statement, it also must have been specified in the IDCAMS DEFINE or ALTER run.

The file name MYFILE must be the same as the file name or ddname specified in your JCL (above).

If you want to use the VSAMCAT feature, you must either include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

For example:

```
FILE MYFILE VSAM OUTPUT FROM TAPEIN REUSE
```

Figure 437 Coding VSAM in the FILE Statement

To create variable-length records, see [Considerations for Processing Variable-Length Records on page 524](#).

The records on the input file must be in sequence by key for this run. In other words, each record written to the output file must have a key higher than the key in the previous record. Duplicates are not allowed.

Entry Sequence VSAM (ESDS) Create

Use VISION:Results statements to define the input file and the file to be created (output). If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code one of the following:

- F Fixed-length records.
- V Variable-length records.
- S Spanned variable-length records.

VSAM records are never blocked.

```
FILE TAPEIN FB 40 4800
FILE MYFILE ESDS OUTPUT FROM TAPEIN F REUSE
```

Figure 438 Coding in the FILE Statement Example 1

The previous example reads each record on the input file and writes it to the output file, unchanged. For information about selecting records, reformatting, or defining fields (data items), see *Advantage VISION:Results for z/OS Getting Started*. For record selection and data manipulation information, see [Chapter 7: Using the IF Command](#) and [Chapter 10: Using the MOVE Command](#).

The keyword REUSE causes any data already in the file (should such be the case) to be replaced with the new data. If REUSE is not specified, all new records are added to the end of any existing data. If REUSE is coded in your FILE statement, it must also have been specified in the IDCAMS DEFINE or ALTER run.

The file name MYFILE must be the same as the file name or ddname specified in your JCL (above).

As each record is added to an ESDS VSAM file, its relative byte address (RBA) is returned to you as either a 4-byte binary or 16-byte packed number. This number can be used as the record key for subsequent random access to the file. You must use RELBYTEX there is any chance that the returned RBA will exceed 4GB in value. If you need this value, code:

```
FILE filename ESDS ... RELBYTE dataname
                        or
FILE filename ESDS ... RELBYTEX dataname
```

Figure 439 Coding in the FILE Statement Example 2

- Dataname contains the RBA of the record written as a 4-byte binary number when RELBYTE is used, or a 16-byte packed number when RELBYTEX is used.
- Dataname can be any data name that is not defined elsewhere in your program.

If you want to use the VSAMCAT feature, you must either include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

```
FILE MYFILE VSAM OUTPUT FROM TAPEIN REUSE
```

Figure 440 Coding in the FILE Statement Example 3

To create variable-length records, see [Considerations for Processing Variable-Length Records on page 524](#).

Relative Record VSAM (RRDS) Create

Use VISION:Results statements to define the input file and the file to be created (output). You also need one or more procedural statements to set up the relative record number (or slot number) of each output record.

If you have not specified `OPTION VSAMCAT` in your program or the keyword `VSAMCAT=Y` in the `DYLINKSTL` macro and the JCL does not include the `DD` or `DLBL` statement for the file, you must code `F` (for fixed-length records). VSAM records are never blocked.

```
FILE TAPEIN FB 40 4800
  BIN NUMBER 4 17 PD
FILE MYFILE RRDS OUTPUT FROM TAPEIN F REUSE RELREC BINNUM
MOVE BIN_NUMBER TO BINNUM
```

Figure 441 Relative Record VSAM (RRDS)

The `RELREC` keyword is required. It must be followed by a data name not defined elsewhere in the program. The value of the data name becomes the relative record number of any given output record. Each record written to the file must have a relative record number higher than the previous record. Duplicates are not allowed.

The example above reads each record on the input file and copies it to the output file, unchanged, using the bin number value in each input record to establish the relative record number of each output record. For information about selecting records, reformatting, or defining fields (data items), see *Advantage VISION:Results for z/OS Getting Started*. For record selection and data manipulation information, see [Chapter 7: Using the IF Command](#) and [Chapter 10: Using the MOVE Command](#).

The keyword `REUSE` causes any data already in the file (should such be the case) to be replaced by the new data. If `REUSE` is not specified, the new input records are inserted among the existing records according to relative record number. The number of a new record must not match that of an existing one. If `REUSE` is coded in your `FILE` statement, it must also have been specified in the `IDCAMS DEFINE` or `ALTER` run.

The file name `MYFILE` must be the same as the file name or ddname specified in your JCL (above).

If you want to use the `VSAMCAT` feature, you must either include `OPTION VSAMCAT` in your program or set the `DYLINKSTL` macro keyword `VSAMCAT` to `Y`, and code `VSAM` in the `FILE` statement. If you include the `DD` or `DLBL` statement for the file in the JCL, then specifying the file type and its characteristics is optional.

For example:

```
FILE MYFILE VSAM OUTPUT FROM TAPEIN REUSE RELREC BINNUM
```

Figure 442 Coding VSAM Output in the FILE Statement

Sequential Read

This section describes reading a VSAM file starting with the first record and reading serially until end of file occurs or a STOP command is issued by the VISION:Results program.

To read a file, you need to describe the file to the operating system using JCL.

```
(VSE)      // ASSGN SYSnnn,X'cuu'  
           // DLBL MYFILE,'MY.FILE.IDENT',    ...  
           // EXTENT SYSnnn,    ...  
  
(z/OS) //MYFILE DD DSN=MY.FILE.IDENT,DISP=SHR
```

Figure 443 Reading a File JCL

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

For the statements used to read VSAM files, see [Key Sequence VSAM \(KSDS\) Sequential Read on page 490](#), [Entry Sequence VSAM \(ESDS\) Sequential Read on page 491](#), and [Relative Record VSAM \(RRDS\) Sequential Read on page 492](#).

Key Sequence VSAM (KSDS) Sequential Read

Use a FILE statement to define the input file. If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code one of the following:

F Fixed-length records.
V Variable-length records.
S Spanned variable-length records.

VSAM records are never blocked.

```
FILE MYFILE KSDS F 40
```

Figure 444 Key Sequence VSAM (KSDS)

To read variable-length records, see [Considerations for Processing Variable-Length Records on page 524](#).

The previous example reads each record on the input file in ascending order by key and makes the record available to any procedure logic that follows. For information about record selection and printing, data handling and definition, and reporting, see *Advantage VISION:Results for z/OS Getting Started* as well as [Chapter 7: Using the IF Command](#), [Chapter 10: Using the MOVE Command](#), and [Chapter 16: Using Report Statements](#).

The file name MYFILE must be the same as the file name or ddname specified in the previous JCL.

If you want to use the VSAMCAT feature, you must either include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

```
FILE MYFILE VSAM
```

Figure 445 Key Sequence VSAM (KSDS) Option

Entry Sequence VSAM (ESDS) Sequential Read

Use a FILE statement to define the input file. If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code one of the following:

- F Fixed-length records.
- V Variable-length records.
- S Spanned variable-length records.

VSAM records are never blocked.

```
FILE MYFILE ESDS F 40
```

Figure 446 Entry Sequence VSAM (ESDS)

To read variable-length records, see [Considerations for Processing Variable-Length Records on page 524](#).

The previous example reads each record on the input file and makes the record available to any procedure logic that follows. Record selection and printing, data handling and definition, and reporting are described in *Advantage VISION:Results for z/OS Getting Started* as well as [Chapter 7: Using the IF Command](#), [Chapter 10: Using the MOVE Command](#), and [Chapter 16: Using Report Statements](#).

The file name MYFILE must be the same as the file name or ddname specified in the previous JCL.

If you want to use the VSAMCAT feature, you must either include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

For example:

```
FILE MYFILE VSAM
```

Figure 447 Entry Sequence VSAM (ESDS) Option

As each record is read from an ESDS VSAM file, its relative byte address (RBA) is returned to you as either a 4-byte binary or 16-byte packed number. This number can be used as the record address for subsequent random access to the file. You must use RELBYTEX if there is any chance that the returned RBA will exceed 46 bytes in value. If you need this value, code:

```
FILE filename ESDS ... RELBYTE dataname  
or  
FILE filename ESDS ... RELBYTEX dataname
```

Figure 448 Coding in the FILE Statement Example 2

- Dataname contains the RBA of the record written as a 4-byte binary number when RELBYTE is used, or a 16-byte packed number when RELBYTEX is used.
- Dataname can be any data name that is not defined elsewhere in your program.

Relative Record VSAM (RRDS) Sequential Read

Use a FILE statement to define the input file. If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code an F (for fixed-length records). VSAM records are never blocked.

```
FILE MYFILE RRDS F 40
```

Figure 449 Relative Record VSAM (RRDS)

The previous example reads each record on the input file in ascending order by relative record number and makes the record available to any procedure logic that follows. Record selection and printing, data handling and definition, and reporting are described in *Advantage VISION:Results for z/OS Getting Started* as well as [Chapter 7: Using the IF Command](#), [Chapter 10: Using the MOVE Command](#), and [Chapter 16: Using Report Statements](#).

The file name MYFILE must be the same as the file name or ddname specified in the previous JCL.

If you want to use the VSAMCAT feature, you must either include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

For example:

```
FILE MYFILE VSAM
```

Figure 450 Coding VSAM in the FILE Statement

As each record is read from an RRDS VSAM file, its relative record number is returned to you. This 4-byte binary number can be used as the record key for subsequent random access to the file. If you need this value, code:

```
FILE filename RRDS ... RELREC recnum
```

Figure 451 Relative Record Number of RRDS VSAM File

- Recnum contains the relative record number of the record read.
- Recnum can be any data name that is not defined elsewhere in your program.

Limited (Skip) Sequential Read

This section describes reading a VSAM file by skipping to a particular record in the file and reading records serially from that point until some condition is met or end of file is encountered. In either case, another similar retrieval then can be initiated, and so on, as often as needed. You must use a READ command to cause the file to be read, and must code a STOP statement to end the run unless your program has a SORT.

To read a file, describe the file to the operating system using JCL.

```
(VSE) // ASSGN SYSnnn,X'cuu'
      // DLBL MYFILE,'MY.FILE.IDENT', ...
      // EXTENT SYSnnn, ...

(z/OS) //MYFILE DD DSN=MY.FILE.IDENT,DISP=SHR
```

Figure 452 Reading a VSAM File by Skipping to a Particular Record

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

For the statements used to read VSAM files in skip sequential mode, see [Key Sequence VSAM \(KSDS\) Skip Sequential Read on page 493](#), [Entry Sequence VSAM \(ESDS\) Skip Sequential Read on page 496](#), and [Relative Record VSAM \(RRDS\) Skip Sequential Read on page 499](#).

Key Sequence VSAM (KSDS) Skip Sequential Read

Use a FILE statement to define the input file. If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code one of the following:

- F Fixed-length records.
- V Variable-length records.
- S Spanned variable-length records.

VSAM records are never blocked.

```
FILE MYFILE KSDS SKIP SEARCHKEY STATUS STATFLG
  F 40 KEYLEN 5
  POSITION POSCODE PARTKEY 3 (optional)
```

Figure 453 VSAM (KSDS) Skip Sequential Read Without OPTION VSAMCAT

If you want to use the VSAMCAT feature, you must either include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

For example:

```
FILE MYFILE VSAM SKIP SEARCHKEY STATUS STATFLG
  POSITION POSCODE PARTKEY 3 (optional)
```

Figure 454 VSAM (KSDS) Skip Sequential Read with OPTION VSAMCAT

The following keywords are required:

SKIP dataname Any name not defined elsewhere in your program. Its length is equal to the key length. It is used to hold the key of the record to be searched for. It is assumed to have a data type of CH (character). If your key is signed numeric, packed decimal, or binary, you must code NU, PD, or BI, respectively, after dataname:

```
SKIP dataname PD
```

Also, if the number of decimal places is other than zero, code it after the data type:

```
SKIP dataname PD 2
```

Note: Reading in a skip sequential file always occurs in a forward manner. However, skipping to a new position within that file can occur forwards and backwards from the current record position. The POSITION dataname can be blank, G, or E.

STATUS dataname Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file:

blank No record found.

Y Record found.

E End of file.

I Duplicate or invalid key.

The following keyword is optional if you specified OPTION VSAMCAT in your program or the VSAMCAT=Y keyword in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file:

KEYLEN nnn Specifies the length of the record key. The allowable values for nnn are 1 to 255.

The following keywords are optional:

POSITION dataname Any name not defined elsewhere in your program. It is a 1-byte field that can be used to control the type of search (positioning) as follows:

blank / G Start reading at the first record whose key is equal to or greater than the value in the SKIP dataname.

A blank is reset to a G after the next read.

E Start reading at the record whose key is equal to the value in the SKIP dataname.

R Repeat the previous positioning.

VISION:Results recognizes the need to skip to a new starting point in the file by comparing the contents of the SKIP dataname with its previous value. To restart the process with the same SKIP key as the previous time (if required), move an R to the POSITION dataname. The R is reset to the previous POSITION dataname (E or G) after the next read.

PARTKEY nnn nnn is the length of the partial (generic) key. It can be any value between 1 and 254, but it must be less than the full key length (KEYLEN).

Code procedure logic to position the file based on a starting key and read records until some condition is met. For example:

```
FILE MYFILE KSDS F 40
  KEYLEN 5  SKIP SEARCHKEY
  STATUS STATFLG
  VSAMREC 40
  VSAMKEY 51

FILE INKEYS F 80
  STATUS INKEYFLG
  STARTKEY 51
  ENDKEY 56
.
NXTGROUP:
  READ INKEYS
  IF INKEYFLG EQ 'E' STOP ENDIF
  MOVE STARTKEY TO SEARCHKEY
  MOVE 'G' TO POSCODE (optional)
READLOOP:
  READ MYFILE
  IF STATFLG NE 'Y' GOTO NXTGROUP ENDIF
  IF VSAMKEY GT ENDKEY GOTO NXTGROUP ENDIF
  HEXPRINT VSAMREC
  GOTO READLOOP
```

Figure 455 Code Procedure Logic Example (KSDS)

Each record in a file INKEYS contains a STARTKEY and an ENDKEY. The program initializes the SKIP dataname (SEARCHKEY) and reads the first record on the VSAM file whose key is equal to or greater than the starting key. It then checks to make sure that a record was found. Finally, the key of the record just read is compared to the highest key required (ENDKEY). All records on the file whose keys fall within the limits are printed, and the program returns to read the next INKEYS record.

When the INKEYS file reaches the end, the program issues a STOP command. You must always tell VISION:Results when to end the run if a skip sequential file is used.

To read variable-length records, see [Considerations for Processing Variable-Length Records on page 524](#).

The file name MYFILE must be the same as the file name or ddname specified in the previous JCL.

Generic Search

A generic search involves using a partial key to point to the group of records required. For example, the first 3 bytes of your key contain a department number and you want to obtain all the records for Department 341. Just add the PARTKEY nnn clause to the FILE statement and move the department number to the SKIP dataname.

KSDS is shown in **bold** and is optional when using the OPTION VSAMCAT statement or the VSAMCAT=Y keyword in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file. Type VSAM in the FILE statement.

```
FILE MYFILE KSDS ... PARTKEY 3
.
WORKAREA
.
DEPT341 3 VALUE '341'
.
MOVE DEPT341 TO SEARCHKEY LENGTH 3
READLOOP:
  READ MYFILE etc.
.
.
GOTO READLOOP
```

Figure 456 PARTKEY nnn Clause Added to the FILE Statement

VISION:Results starts reading records at the first one whose key is equal to or greater than 341, followed by all binary 00s.

Entry Sequence VSAM (ESDS) Skip Sequential Read

Use a FILE statement to define the input file. If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code one of the following:

- F Fixed-length records.
- V Variable-length records.
- S Spanned variable-length records.

VSAM records are never blocked.

```
FILE MYFILE ESDS SKIP SEARCHRBA STATUS STATFLG
RELBYTE ESDSRBA F 40
POSITION POSCODE (optional)
```

Figure 457 VSAM (ESDS) Skip Sequential Read Without OPTION VSAMCAT

If you want to use the VSAMCAT feature, you must either include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

```
FILE MYFILE VSAM SKIP SEARCHRBA STATUS STATFLG
RELBYTE ESDSRBA
POSITION POSCODE (optional)
```

Figure 458 VSAM (ESDS) Skip Sequential Read with OPTION VSAMCAT

The following keywords are required:

SKIP dataname	Any name not defined elsewhere in your program. Its length is always 4. It holds the relative byte address (RBA) of the record to be searched for. It cannot be used with SKIPX or RELBYTEX
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: blank Record not found. Y Record found. E End of file. I Duplicate or invalid key.
RELBYTE dataname	Any name not defined elsewhere in your program. Its length is always 4. After a READ, the system returns the 4-byte binary RBA of the record just read in dataname. It cannot be used with RELBYTEX or SKIPX.
RELBYTEX dataname	Any name not defined elsewhere in your program. Its length is always 16. After a READ, the system returns the 16-byte packed RBA of the record just read in dataname. Use this keyword instead of RELBYTE if the VSAM file is greater than 4GB. It cannot be used with RELBYTE or SKIP.
SKIPX dataname	Any name not defined elsewhere in your program. Its length is always 16. It holds the extended relative byte address (xRBA) of the record to be searched for. Use this keyword instead of SKIP if the VSAM file is greater than 4GB. It cannot be used with RELBYTE or SKIP.

The following keyword is optional:

POSITION dataname	Any name not defined elsewhere in your program. It is a 1-byte field that can be used to control the type of search (positioning) as follows:
E (default)	Start reading at the record whose RBA is equal to the value in dataname.
R	Repeat the previous positioning.

VISION:Results recognizes the need to skip to a new starting point in the file by comparing the contents of the SKIP or SKIPX dataname with its previous value. To restart the process with the same RBA as the previous time (if required), move an R to the POSITION dataname.

Code procedure logic to position the file based on a starting key and read records until some condition is met. For example:

```
FILE MYFILE ESDS F 40
  SKIP SEARCHRBA
  STATUS STATFLG
  RELBYTE ESDSRBA
ESDSREC 40

FILE INKEYS F 80
  STATUS INKEYFLG
  STARTRBA 4 1 BI
  ENDRBA   4 5 BI
.
.
NXTGROUP:
  READ INKEYS
  IF INKEYFLG EQ 'E' STOP ENDIF
  MOVE STARTRBA TO SEARCHRBA
READLOOP:
  READ MYFILE
  IF STATFLG NE 'Y' GOTO NXTGROUP ENDIF
  IF ESDSRBA GT ENDRBA GOTO NXTGROUP ENDIF
  MOVE ESDSRBA TO DYLPTCOMM
  HEXPRINT ESDSREC
  GOTO READLOOP
```

Figure 459 Code Procedure Logic Example (ESDS)

In the previous example, each record in a file INKEYS contains a STARTRBA and an ENDRBA. The program initializes the SKIP dataname (SEARCHRBA) and reads the record on the VSAM file that has an RBA equal to the starting RBA. It then checks to make sure that the record was found. Finally, the RBA of the record just read is compared to the highest RBA required (ENDRBA). All records on the file whose RBAs fall within the limits are printed (together with their RBAs), and the program returns to read the next INKEYS record.

When the INKEYS file reaches the end, the program issues a STOP command. It always is necessary to tell VISION:Results when to end the run if a skip sequential file is used.

To read variable-length records, see [Considerations for Processing Variable-Length Records on page 524](#).

The file name MYFILE must be the same as the file name or ddname specified in the previous JCL.

Relative Record VSAM (RRDS) Skip Sequential Read

Use a FILE statement to define the input file. If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code an F (for fixed-length records). VSAM records are never blocked.

```
FILE MYFILE RRDS F 40 SKIP SEARCHKEY STATUS STATFLG
  POSITION POSCODE RELREC RECNUM (optional)
```

Figure 460 VSAM (RRDS) Skip Sequential Read Without OPTION VSAMCAT

If you want to use the VSAMCAT feature, you must either include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

```
FILE MYFILE VSAM SKIP SEARCHKEY STATUS STATEFLG
  POSITION POSCODE RELREC RECNUM (optional)
```

Figure 461 VSAM (RRDS) Skip Sequential Read with OPTION VSAMCAT

The following keywords are required:

SKIP dataname	Any name not defined elsewhere in your program. Its length is always 4. It holds the relative record number of the record to be searched for.								
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: <table> <tr> <td>blank</td><td>No record found.</td></tr> <tr> <td>Y</td><td>Record found.</td></tr> <tr> <td>E</td><td>End of file.</td></tr> <tr> <td>I</td><td>Duplicate or invalid key.</td></tr> </table>	blank	No record found.	Y	Record found.	E	End of file.	I	Duplicate or invalid key.
blank	No record found.								
Y	Record found.								
E	End of file.								
I	Duplicate or invalid key.								

The following keywords are optional:

POSITION dataname	Any name not defined elsewhere in your program. It is a 1-byte field that can be used to control the type of search (positioning) as follows:
blank /	Start reading at the first record in which the
G	relative record number is equal to or greater than the value in the SKIP dataname.
E	Start reading at the record whose relative record number is equal to the value in the SKIP dataname.
R	Repeat the previous positioning.

VISION:Results recognizes the need to skip to a new starting point in the file by comparing the contents of the SKIP dataname with its previous value. Therefore, to restart the process with the same SKIP key as the previous time (if required), you would need to move an R to the POSITION dataname.

RELREC dataname Relative record number of the record read, if positioning was successful.

Code procedure logic to position the file based on a starting record number and read records until some condition is met. For example:

```
FILE MYFILE RRDS F 40
  SKIP SEARCHKEY
  STATUS STATFLG POSITION POSCODE
  RELREC RECNUM
  VSAMREC 40

FILE INKEYS F 80 STATUS INKEYFLG
  STARTNUM 4 BI
  ENDNUM 4 BI
.
.
NXTGROUP:
  READ INKEYS
  IF INKEYFLG EQ 'E' STOP ENDIF
  MOVE STARTNUM TO SEARCHKEY
  MOVE 'G' TO POSCODE (optional)
READLOOP:
  READ MYFILE
  IF STATFLG NE 'Y' GOTO NXTGROUP ENDIF
  IF RECNUM GT ENDNUM
  GOTO NXTGROUP ENDIF
  PRINT VSAMREC
  GOTO READLOOP
```

Figure 462 Code Procedure Logic Example (RRDS)

In the previous example, each record in a file INKEYS contains a STARTNUM and an ENDNUM. The program initializes the SKIP dataname (SEARCHKEY) and reads the first record on the VSAM file whose key is equal to or greater than the starting key. It then checks to make sure that a record was found. Finally, the key of the record just read is compared to the highest key required (ENDNUM). All records on the file whose keys fall within the limits are printed, and the program returns to read the next INKEYS record.

When the INKEYS file reaches the end, the program issues a STOP command. It is always necessary to tell VISION:Results when to end the run if a skip sequential file is used.

The file name MYFILE must be the same as the file name or ddname specified in the previous JCL.

Random Read

This section describes reading records, one at a time and in no particular sequence, from a VSAM file. The program must identify precisely which record is wanted prior to issuing the READ request. After the READ, one of two conditions exists: either the requested record is found, or it is not found. If the record is not found, the program must not attempt to access the record area because it may not have been allocated.

To read a file, describe the file to the operating system using JCL.

```
(VSE)      // ASSGN SYSnnn,X'cuu'
           // DLBL MYFILE,'MY.FILE.IDENT',    ...
           // EXTENT SYSnnn,    ...

(z/OS) //MYFILE DD DSN=MY.FILE.IDENT,DISP=SHR
```

Figure 463 Random Reading a File Using JCL

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

For the statements used to read VSAM files in random mode, see [Key Sequence VSAM \(KSDS\) Random Read on page 501](#), [Entry Sequence VSAM \(ESDS\) Random Read on page 503](#), and [Relative Record VSAM \(RRDS\) Random Read on page 505](#).

Key Sequence VSAM (KSDS) Random Read

Use a FILE statement to define the input file. If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code one of the following:

- F Fixed-length records.
- V Variable-length records.
- S Spanned variable-length records.

VSAM records are never blocked.

```
FILE MYFILE KSDS RANDOM SEARCHKEY STATUS STATFLG
  F 40 KEYLEN 5
```

Figure 464 Key Sequence VSAM (KSDS) Random Read Without OPTION VSAMCAT

If you want to use the VSAMCAT feature, you must either include `OPTION VSAMCAT` in your program or set the `DYLINSTL` macro keyword `VSAMCAT` to `Y`, and code `VSAM` in the `FILE` statement. If you include the `DD` or `DLBL` statement for the file in the `JCL`, then specifying the file type and its characteristics is optional.

For example:

```
FILE MYFILE VSAM RANDOM SEARCHKEY STATUS STATFLG
```

Figure 465 Key Sequence VSAM (KSDS) Random Read with `OPTION VSAMCAT`

The following keywords are required:

RANDOM dataname Any name not defined elsewhere in your program. Its length is equal to the key length. It holds the key of the record to be searched for. It is assumed to have a data type of `CH` (character). If your key is signed numeric, packed decimal, or binary, you must code `NU`, `PD`, or `BI`, respectively, after dataname:

```
RANDOM dataname PD
```

Also, if the number of decimal places is other than zero, code it after the data type:

```
RANDOM dataname PD 2
```

STATUS dataname Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a `READ` has been issued for this file:

blank No record found.

Y Record found.

E End of file.

I Invalid or duplicate key.

The following keyword is optional if you specified `OPTION VSAMCAT` in your program or the `VSAMCAT=Y` keyword in the `DYLINSTL` macro and the `JCL` does not include the `DD` or `DLBL` statement for the file:

KEYLEN nnn Specifies the length of the record key. The allowable values for `nnn` are 1 to 255.

Code procedure logic to set up the search key, read, and check the result:

```
FILE MYFILE KSDS F 40
  KEYLEN 5
  RANDOM SEARCHKEY
  STATUS STATFLG
VSAMREC 40
```

```
FILE TAPEIN FB 80 3200
  STATUS TAPESTAT
EMPLOYEE 5
.
```

Figure 466 Code Procedure Logic Example (KSDS) (Page 1 of 2)

```

READLOOP:
  READ TAPEIN
  IF TAPESTAT EQ 'E' STOP ENDIF
  MOVE EMPLOYEE TO SEARCHKEY
  READ MYFILE
  IF STATFLG NE 'Y' GOTO NOTFOUND ENDIF
  HEXPRINT VSAMREC
  GOTO READLOOP

```

Figure 466 Code Procedure Logic Example (KSDS) (Page 2 of 2)

In the previous example, each record in a file TAPEIN contains an employee number (EMPLOYEE). The program moves this number to the RANDOM dataname as a key and issues a READ for the VSAM file. If the record is not found, the program branches to an error routine. If the record is found, the record prints and the program returns to read another TAPEIN record.

When the TAPEIN file reaches the end, the program issues a STOP command. It is always necessary to tell VISION:Results when to end the run if a RANDOM read is used.

To read variable-length records, see [Considerations for Processing Variable-Length Records on page 524](#).

The file name MYFILE must be the same as the file name or ddname specified in the previous JCL.

Entry Sequence VSAM (ESDS) Random Read

Use a FILE statement to define the input file. If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code one of the following:

- F Fixed-length records.
- V Variable-length records.
- S Spanned variable-length records.

VSAM records are never blocked.

```
FILE MYFILE ESDS RANDOM SEARCHRBA STATUS STATFLG F 40
```

Figure 467 Entry Sequence VSAM (ESDS) Random Read Without OPTION VSAMCAT

If you want to use the VSAMCAT feature, you must either include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

For example:

```
FILE MYFILE VSAM RANDOM SEARCHRBA STATUS STATFLG
```

Figure 468 Entry Sequence VSAM (ESDS) Random Read with OPTION VSAMCAT

The following keywords are required:

RANDOM dataname	Any name not defined elsewhere in your program. Its length is always 4. It holds the relative byte address (RBA) of the record to be searched for. It cannot be used with RANDOMX.
RANDOMX dataname	Any name not defined elsewhere in your program. Its length is always 16. It holds the relative byte address (RBA) of the record to be searched for. Use this keyword instead of RANDOMX if the VSAM file is greater than 4GB. It cannot be used with RANDOM.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: blank Record not found. Y Record found. E End of file. I Duplicate or invalid key.

For fixed-length records, the relative byte address (RBA) of any given record number can be calculated. The RBA of any record is the sum of the lengths of all preceding records. For fixed-length records, the RBA of any record *n* is found as follows:

$$\text{RBA} = (n - 1) * \text{length}$$

Code procedure logic to set up the search key, read, and check the result:

```
FILE MYFILE ESDS F 40
  RANDOM SEARCHRBA
  STATUS STATFLG
VSAMREC 40

FILE TAPEIN FB 80 3200
  STATUS TAPESTAT
RECNUM 4 BI
.
.
READLOOP:
  READ TAPEIN
  IF TAPESTAT EQ 'E' STOP ENDIF
  SEARCHRBA = (RECNUM - 1) * 40
  READ MYFILE
  IF STATFLG NE 'Y' GOTO NOTFOUND ENDIF
  HEXPRINT VSAMREC
  GOTO READLOOP
.
.
```

Figure 469 Code Procedure Logic Example (ESDS)

In the previous example, each record in a file TAPEIN contains the record number (RECNUM) of a related record in an ESDS file. The ESDS file has fixed-length, 40-byte records. The program calculates the RBA of the required record, places the result in the RANDOM dataname (SEARCHRBA), and issues a READ for the VSAM file. If the record is not found, the program branches to an error routine. If the record is found, it is printed and the program returns to read another TAPEIN record.

When the TAPEIN file reaches the end, the program issues a STOP command. It is always necessary to tell VISION:Results when to end the run if a RANDOM or RANDOMX read is used.

To read variable-length records, see [Considerations for Processing Variable-Length Records on page 524](#).

The file name MYFILE must be the same as the file name or ddname specified in the previous JCL.

Relative Record VSAM (RRDS) Random Read

Use a FILE statement to define the input file. If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code an F (for fixed-length records). VSAM records are never blocked.

```
FILE MYFILE RRDS F 40 RANDOM SEARCHKEY STATUS STATFLG
```

Figure 470 Relative Record VSAM (RRDS) Random Read Without OPTION VSAMCAT

If you want to use the VSAMCAT feature, you either must include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

```
FILE MYFILE VSAM RANDOM SEARCHKEY STATUS STATFLG
```

Figure 471 Relative Record VSAM (RRDS) Random Read with OPTION VSAMCAT

The following keywords are required:

RANDOM dataname	Any name not defined elsewhere in your program. Its length is always 4. It holds the relative record number of the record to be searched for.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: blank No record found. Y Record found. E End of file. I Duplicate or invalid key.

Code procedure logic to set up the search key, read, and check the result:

```
FILE MYFILE RRDS F 40
  RANDOM SEARCHKEY
  STATUS STATFLG
VSAMREC 40

FILE TAPEIN FB 80 3200
  STATUS TAPESTAT
SLOTNUM 4 BI
.
.
READLOOP:
  READ TAPEIN
  IF TAPESTAT EQ 'E' STOP ENDIF
  MOVE SLOTNUM TO SEARCHKEY
  READ MYFILE
  IF STATFLG NE 'Y' GOTO NOTFOUND ENDIF
  HEXPRINT VSAMREC
  GOTO READLOOP
.
.
```

Figure 472 Code Procedure Logic Example (RRDS)

In the previous example, each record in a file TAPEIN contains a slot number (SLOTNUM). The program moves this number to the RANDOM dataname as a key and issues a READ for the VSAM file. If the record is not found, the program branches to an error routine. If the record is found, it is printed and the program returns to read another TAPEIN record.

When the TAPEIN file reaches the end, the program issues a STOP command. It is always necessary to tell VISION:Results when to end the run if a RANDOM read is used.

The file name MYFILE must be the same as the file name or ddname specified in the previous JCL.

Sequential Update

This section describes reading each record in a VSAM file, starting at the beginning, and updating the file in place (that is, without copying the old file to a new file). You must use a READ command to cause the file to be read, and a STOP command must be coded to end the run, unless your program has a SORT.

Note: This type of run is not restartable. Once one record has been added, changed, or deleted, the file is not the same, regardless of why the job ended. Create a method to recover the original file, as a precaution, before running a program that uses this technique.

For KSDS and RRDS VSAM files, you can add (insert) new records, change existing records, and delete (physically remove) existing records.

For ESDS VSAM files, you can change existing records and add new records to the end of the file only. You cannot change the length of an existing record. Records can be logically deleted by setting a flag of your choice in the record, checked by subsequent application programs to determine the status of the record. You can remove deleted records by selectively copying the file to another file.

To read a file, describe the file to the operating system using JCL.

```
(VSE)      // ASSGN SYSnnn,X'cuu'
           // DLBL MSTRFL,'MY.FILE.IDENT',    ...
           // EXTENT SYSnnn,    ...

(z/OS)    //MSTRFL DD DSN=MY.FILE.IDENT,DISP=OLD
```

Figure 473 Reading a File Using JCL

Note the disposition of OLD in the previous example. This choice (also available under VSE, Levels 2 and up) prevents other users from accessing the file while your program is updating it.

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

For the statements needed to sequentially update VSAM files, see [Key Sequence VSAM \(KSDS\) Sequential Update on page 508](#), [Entry Sequence VSAM \(ESDS\) Sequential Update on page 509](#), and [Using the RELBYTE or RELBYTEX dataname on page 511](#).

Key Sequence VSAM (KSDS) Sequential Update

If you have not specified `OPTION VSAMCAT` in your program or the keyword `VSAMCAT=Y` in the `DYLNSTL` macro and the JCL does not include the `DD` or `DLBL` statement for the file, you must code one of the following:

- F** Fixed-length records.
- V** Variable-length records.
- S** Spanned variable-length records.

VSAM records are never blocked.

```
FILE MSTRFL KSDS IO F 40 KEYLEN 5
      ERASE ERASEFLG STATUS STATFLG
MASTREC 40 MASTEMPL 5 2
```

Figure 474 Key Sequence VSAM (KSDS) Sequential Update Without `OPTION VSAMCAT`

If you want to use the `VSAMCAT` feature, you must either include `OPTION VSAMCAT` in your program or set the `DYLNSTL` macro keyword `VSAMCAT` to `Y`, and code `VSAM` in the `FILE` statement. If you include the `DD` or `DLBL` statement for the file in the JCL, then specifying the file type and its characteristics is optional.

```
FILE MSTRFL VSAM IO
      ERASE ERASEFLG STATUS STATFLG
MASTREC 40 MASTEMPL 5 2
```

Figure 475 Key Sequence VSAM (KSDS) Sequential Update with `OPTION VSAMCAT`

The following keywords are required:

- IO** Allows the file to be updated in place. The record format and record size must be specified on the `FILE` statement in conjunction with this keyword, unless `VSAMCAT` is in effect and `VSAM` is coded in the `FILE` statement.

The following keyword is optional if you specified `OPTION VSAMCAT` in your program or the `VSAMCAT=Y` keyword in the `DYLNSTL` macro and the JCL does not include the `DD` or `DLBL` statement for the file:

- KEYLEN nnn** Specifies the length of the record key. The allowable values for `nnn` are 1 to 255.

The following keywords are optional:

- ERASE dataname** Any name not defined elsewhere in your program. It is a 1-byte field that should contain an `E` to cause a `WRITE` command to delete the record just read.
- STATUS dataname** Any name not defined elsewhere in your program. It is a 1-byte field that contains an `E` when end of file occurs.

Suppose that you have a KSDS employee master file and want to implement a flat 9% raise for all employees hired before Oct. 15, 1995. In addition, you want to delete the master record of any employee who was terminated before Apr. 15, 1995.

The following procedure logic can be coded:

```
FILE MSTRFL RSDS IO F 40
  KEYLEN 5
  ERASE ERASEFLG
  STATUS STATFLG
MASTREC 40
MASTEMPL 5
HIRE_DATE 8 7
TERM_DATE 8 15
RATE 2 23 RI
.
READMSTR:
  READ MSTRFL
  IF MASTFLAG EQ 'E' STOP ENDIF
COMPARE:
  IF (TERM_DATE NE 0 AND TERM_DATE LT 19950415)
    MOVE 'E' TO ERASEFLG GOTO REWRITE ENDIF
  IF (HIRE_DATE LT 19951015 AND TERM_DATE EQ 0)
    RATE = RATE * 1.09 GOTO REWRITE ENDIF
  GOTO READMSTR
REWRITE:
  WRITE MSTRFL
  GOTO READMSTR
```

Figure 476 Code Procedure Logic Example (KSDS)

To insert a record, read the file until you find the first record with a key higher than one to be added. Then, build the new record in the record area and issue a WRITE. Inserting a record causes the file to be positioned at that new record. The next READ of the master file retrieves the same record that caused the original insertion.

When the MSTRFL file reaches the end, the program issues a STOP command. You must specify when to end the run if an IO file is used.

To update variable-length records, see [Considerations for Processing Variable-Length Records on page 524](#).

The file name MSTRFL must be the same as the file name or ddname specified in the previous JCL.

Entry Sequence VSAM (ESDS) Sequential Update

Use a FILE statement to define the input file. If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code one of the following:

- F Fixed-length records.
- V Variable-length records.
- S Spanned variable-length records.

VSAM records are never blocked.

```
FILE MSTRFL ESDS IO F 40 STATUS STATFLG RELBYTE MASTRBA
```

Figure 477 Entry Sequence VSAM (ESDS) Sequential Update Without OPTION VSAMCAT

If you want to use the VSAMCAT feature, you must either include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

```
FILE MSTRFL VSAM IO STATUS STATFLG RELBYTE MASTRBA
```

Figure 478 Entry Sequence VSAM (ESDS) Sequential Update with OPTION VSAMCAT

The following keyword is required:

IO	Allows the file to be updated in place. Record format and record size must be specified on the FILE statement in conjunction with this keyword, unless VSAMCAT is in effect and VSAM is coded in the FILE statement.

The following keywords are optional:

RELBYTE dataname	Any name not defined elsewhere in your program. Its length is always 4. After a READ, the system returns the 4-byte binary, relative byte address (RBA) of the record just read in dataname. It cannot be used with RELBYTEX.
RELBYTEX dataname	Any name not defined elsewhere in your program. Its length is always 16. After a READ, the system returns the 16-byte packed, relative byte address (RBA) of the record just read in dataname. Use this keyword instead of RELBYTE if the VSAM file is greater than 4GB. It cannot be used with RELBYTE.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains an E when end of file occurs.

Suppose that certain records on an ESDS file are missing a date. You need to find those records and substitute today's date. The following procedure logic can be coded:

```
FILE MSTRFL ESDS IO F 40
  STATUS STATFLG
  RELBYTE MASTRBA
MASTREC 40
DATEINFO 8 25
.
.
READFILE:
  READ MSTRFL
  IF STATFLG EQ 'E' STOP ENDIF
  IF DATEINFO NE '00/00/00'
    GOTO READFILE  ENDIF
  MOVE DYLDATA TO DATEINFO
  WRITE MSTRFL
  GOTO READFILE
```

Figure 479 Code Procedure Logic Example (ESDS)

When the MSTRFL file reaches the end, the program issues a STOP command. You must specify when to end the run if an IO file is used.

To update variable-length records, see [Considerations for Processing Variable-Length Records on page 524](#).

The file name MSTRFL must be the same as the file name or ddname specified in the previous JCL.

Using the RELBYTE or RELBYTEX dataname

If the records on the master file are identified by their relative byte address (RBA) on a transaction file, the RELBYTE or RELBYTEX dataname can be used after a READ command to match masters and transactions.

To add a record to the ESDS file after end of file (STATUS dataname = 'E'), move the new record to the ESDS file record area, move X'FFFFFFFF' (or -1) to the RELBYTE dataname, or -1 to the RELBYTEX dataname, and issue the WRITE. This process can be repeated as required.

Relative Record VSAM (RRDS) Sequential Update

Use VISION:Results statements to define the file. If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code an F (for fixed-length records). VSAM records are never blocked.

```
FILE MSTRFL RRDS IO F 40 RELREC RECNUM
  ERASE ERASEFLG STATUS STATFLG
MASTREC 40 MASTEMPL 5 2
```

Figure 480 Relative Record VSAM (RRDS) Sequential Update Without OPTION VSAMCAT

If you want to use the VSAMCAT feature, you must either include `OPTION VSAMCAT` in your program or set the `DYLNSTL` macro keyword `VSAMCAT` to `Y`, and code `VSAM` in the `FILE` statement. If you include the `DD` or `DLBL` statement for the file in the `JCL`, then specifying the file type and its characteristics is optional.

```
FILE MSTRFL VSAM IO RELREC RECNUM  
      ERASE ERASEFLG STATUS STATFLG  
MASTREC 40 MASTEMPL 5 2
```

Figure 481 Relative Record VSAM (RRDS) Sequential Update with `OPTION VSAMCAT`

The following keyword is required:

IO	Allows the file to be updated in place. The record format and record size must be specified on the <code>FILE</code> statement in conjunction with this keyword, unless <code>VSAMCAT</code> is in effect and <code>VSAM</code> is coded in the <code>FILE</code> statement.

The following keywords are optional:

RELREC dataname	Any name not defined elsewhere in your program. It contains the relative record number of the last record read and supplies the number of a new record to be inserted.
ERASE dataname	Any name not defined elsewhere in your program. It is a 1-byte field that should contain an <code>E</code> to cause a <code>WRITE</code> command to delete the record just read.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains an <code>E</code> when end of file occurs.

Suppose that you have an RRDS employee master file and want to implement a flat 9% raise for all employees hired before October 15, 1995. In addition, you want to delete the master record of any employee who was terminated before April 15, 1995.

The following procedure logic can be coded:

```
FILE MSTRFL RRDS IO F 40
  RELBYTE MASTRBA
  ERASE ERASEFLG
  STATUS STATFLG
MASTREC 40
MASTEMPL 5 2
HIRE_DATE 8 7
TERM_DATE 8 15
RATE 2 23 BI
.
.
READMSTR:
  READ MSTRFL
  IF MASTFLAG EQ 'E' STOP ENDIF
COMPARE:
  IF (TERM DATE NE 0 AND TERM DATE LT 19950415)
    MOVE 'E' TO ERASEFLG GOTO REWRITE ENDIF
  IF (HIRE DATE LT 19951015 AND TERM DATE EQ 0)
    RATE = RATE * 1.09 GOTO REWRITE ENDIF
  GOTO READMSTR
REWRITE:
  WRITE MSTRFL
  GOTO READMSTR
```

Figure 482 Code Procedure Logic Example (RRDS)

To insert a record, read the file until you find the first record whose number is higher than the one to be added. Then, build the new record in the record area, move its relative record number to the RELREC dataname, and issue a WRITE. The insertion of a record causes the file to be positioned at that new record. The next READ of the master file retrieves the same record that caused the original insertion.

When the MSTRFL file reaches the end, the program issues a STOP command. You must specify when to end the run if an IO file is used.

The file name MSTRFL must be the same as the file name or ddname specified in your JCL (above).

Random Update

This section describes the random insertion, changing, and deletion of records on a VSAM file. The file is updated in place (that is, without copying the old file to a new file).

Note: This type of run is not restartable. Once one record has been added, changed, or deleted, the file is not the same, regardless of why the job ended. Create a method of recovering the original file, as a precaution, before running a program that uses this technique.

For KSDS and RRDS VSAM files, you can add (insert) new records, change existing records, and delete (physically remove) existing records.

For ESDS VSAM files, you can change existing records and add new records to the end of the file only. You cannot change the length of an existing record. Records can be logically deleted by setting a flag of your choice in the record, checked by subsequent application programs to determine the status of the record. You can remove deleted records by selectively copying the file to another file.

To update a file, you need to describe the file to the operating system using JCL.

```
(VSE)      // ASSGN SYSnnn,X'cuu'  
           // DLBL MSTRFL,'MY.FILE.IDENT',    ...  
           // EXTENT SYSnnn,    ...  
  
(z/OS)    //MSTRFL  DD DSN=MY.FILE.IDENT,DISP=OLD
```

Figure 483 Updating a File Using JCL

Note the disposition of OLD (above). This choice (also available under VSE, Levels 2 and up) prevents other users from accessing the file while your program is updating it.

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

For the statements needed to randomly update VSAM files, see [Key Sequence VSAM \(KSDS\) Random Update on page 514](#), [Entry Sequence VSAM \(ESDS\) Random Update on page 516](#), and [Relative Record VSAM \(RRDS\) Random Update on page 519](#).

Key Sequence VSAM (KSDS) Random Update

Use VISION:Results statements to define the input file. If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code one of the following:

- F Fixed-length records.
- V Variable-length records.
- S Spanned variable-length records.

VSAM records are never blocked.

```
FILE MSTRFL KSDS IO RANDOM SEARCHKEY F 40 KEYLEN 5  
ERASE ERASEFLG STATUS STATFLG  
MASTREC 40
```

Figure 484 Key Sequence VSAM (KSDS) Random Update Without OPTION VSAMCAT

If you want to use the VSAMCAT feature, you either must include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

For example:

```
FILE MSTRFL VSAM IO RANDOM SEARCHKEY
  ERASE ERASEFLG STATUS STATFLG
MASTREC 40
```

Figure 485 Key Sequence VSAM (KSDS) Random Update with OPTION VSAMCAT

The following keywords are required:

IO Allows the file to be updated in place. The record format and record size must be specified on the FILE statement in conjunction with this keyword, unless VSAMCAT is in effect and VSAM is coded in the FILE statement.

RANDOM dataname Any name not defined elsewhere in your program. Its length is equal to the key length. It holds the key of the record to be searched for. It is assumed to have a data type of CH (character). If your key is signed numeric, packed decimal, or binary, you must code NU, PD, or BI, respectively, after dataname:

```
RANDOM dataname PD
```

Also, if the number of decimal places is other than zero, code it after the data type:

```
RANDOM dataname PD 2
```

STATUS dataname Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file:

blank No record found.

Y Record found.

E End of file.

I Duplicate or invalid key.

The following keyword is optional if you specified OPTION VSAMCAT in your program or the VSAMCAT=Y keyword in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file:

KEYLEN nnn Specifies the length of the record key. The allowable values for nnn are 1 to 255.

The following keyword is optional:

ERASE dataname Any name not defined elsewhere in your program. It is a 1-byte field that should contain an E to cause a WRITE command to delete the record just read.

Suppose that a KSDS VSAM file uses employee number as its key. There is also a transaction file with records in no particular sequence that have the same format as records on the VSAM file. If the transaction employee number (EMPLOYEE) is not on the VSAM file, the employee is added. If the employee transaction matches a record on the VSAM file, you want to replace the VSAM record or delete it (if the employee is terminated).

The following procedure logic can be coded:

```
FILE MSTRFL KSDS IO F 40
  KEYLEN 5
  RANDOM SEARCHKEY
  ERASE ERASEFLG
  STATUS STATFLG
  MASTREC 40

FILE TRANS F 40 STATUS TRANSEOF
TRANREC 40
EMPLOYEE 5 1
TERMINATE 1 6
.
.
READTRAN:
  READ TRANS
  IF TRANSEOF EQ 'E' STOP ENDIF
  MOVE EMPLOYEE TO SEARCHKEY
  READ MSTRFL
  IF STATFLG NE 'Y' GOTO ADDRDN ENDIF
  IF TERMINATE EQ 'Y' MOVE 'E' to ERASEFLG
  GOTO REWRITE ENDIF
  MOVE TRANREC TO MASTREC
REWRITE:
  WRITE MSTRFL
  GOTO READTRAN
ADDRDN:
  MOVE TRANREC TO MASTREC
  WRITE MSTRFL
  GOTO READTRAN
```

Figure 486 Code Procedure Logic Example (KSDS)

When the TRANS file reaches the end, the program issues a STOP command. You must specify when to end the run if an IO RANDOM update is used.

Before a record can be replaced, it must be read and found. Before a record can be added, it must be read and not found, even if you know it is not there.

To update variable-length records, see [Considerations for Processing Variable-Length Records on page 524](#).

The file name MSTRFL must be the same as the file name or ddname specified in the previous JCL.

Entry Sequence VSAM (ESDS) Random Update

Use VISION:Results statements to define the input file. If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code one of the following:

- F Fixed-length records.
- V Variable-length records.
- S Spanned variable-length records.

VSAM records are never blocked.

```
FILE MSTRFL ESDS IO RANDOM SEARCHRBA F 40
STATUS STATFLG
MASTREC 40 USEDCCOUNT 4 27 PD
```

Figure 487 Entry Sequence VSAM (ESDS) Random Update Without OPTION VSAMCAT

If you want to use the VSAMCAT feature, you must either include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

```
FILE MSTRFL VSAM IO RANDOM SEARCHRBA
STATUS STATFLG
MASTREC 40 USEDCCOUNT 4 27 PD
```

Figure 488 Entry Sequence VSAM (ESDS) Random Update with OPTION VSAMCAT

The following keywords are required:

IO	Allows the file to be updated in place. The record format and record size must be specified on the FILE statement in conjunction with this keyword, unless VSAMCAT is in effect and VSAM is coded in the FILE statement.								
RANDOM dataname	Any name not defined elsewhere in your program. Its length is always 4. It holds the relative byte address (RBA) of the record to be searched for. It cannot be used with RANDOMX.								
RANDOMX dataname	Any name not defined elsewhere in your program. Its length is always 16. It holds the relative byte address (RBA) of the record to be searched for. Use this keyword instead of RANDOMX if the VSAM file is greater than 4GB. It cannot be used with RANDOM.								
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: <table> <tr> <td>blank</td><td>Record not found.</td></tr> <tr> <td>Y</td><td>Record found.</td></tr> <tr> <td>E</td><td>End of file.</td></tr> <tr> <td>I</td><td>Duplicate or invalid key.</td></tr> </table>	blank	Record not found.	Y	Record found.	E	End of file.	I	Duplicate or invalid key.
blank	Record not found.								
Y	Record found.								
E	End of file.								
I	Duplicate or invalid key.								

Note that for fixed-length records, the relative byte address (RBA) of any given record number can be calculated. The RBA of any record is the sum of the lengths of all preceding records. Therefore, for fixed-length records, the RBA of any record *n* is found as follows:

$$\text{RBA} = (n - 1) * \text{length}$$

Suppose that each record in a file TRANS contains the record number (RECNUM) of a related record in an ESDS file. The ESDS file has fixed-length, 40-byte records. The program adds the item count from the transaction record (ITEMS) to the USEDCount in the master record.

The procedure logic calculates the RBA of the required record, places the result in the RANDOM dataname (SEARCHRBA), and issues a READ for the VSAM file. If the record is not found, the program branches to an error routine. If the record is found, it is updated and the program returns to read another TRANS record.

```
FILE MSTRFL ESDS IO F 40
  RANDOM SEARCHRBA
  STATUS STATFLG
MASTRFC 40
USEDCount 4 27 PD

FILE TRANS F 40 STATUS TRANSEOF
RECNUM 4 BI
.
.
READTRAN:
  READ TRANS
  IF TRANSEOF EQ 'E' STOP ENDIF
  SEARCHRBA = (RECNUM - 1) * 40
  READ MSTRFL
  IF STATFLG NE 'Y' GOTO KEYERROR ENDIF
  USEDCount = USEDCount + ITEMS
  WRITE MSTRFL
  GOTO READTRAN
.
.
```

Figure 489 Code Procedure Logic Example (ESDS)

When the TRANS file reaches the end, the program issues a STOP command. You must specify when to end the run if an IO RANDOM read is used.

To update variable-length records, see [Considerations for Processing Variable-Length Records on page 524](#). Remember that you cannot change the length of a record in an ESDS file.

The file name MSTRFL must be the same as the file name or ddname specified in your JCL ([Figure 489](#)).

Using the RANDOM/RANDOMX dataname

When performing Random Update processing, if you need to add a record to the ESDS file after end of file (STATUS dataname is E), move the new record to the ESDS record area. Then move X'FFFFFFFF' or -1 to the RANDOM dataname, or move -1 to the RANDOMX dataname, and issue the WRITE. This process can be repeated as required.

Relative Record VSAM (RRDS) Random Update

Use VISION:Results statements to define the input file. If you have not specified `OPTION VSAMCAT` in your program or the keyword `VSAMCAT=Y` in the `DYLISTL` macro and the JCL does not include the `DD` or `DLBL` statement for the file, you must code an `F` (for fixed-length records). VSAM records are never blocked.

```
FILE MSTRFL RRDS F 40 IO RANDOM SEARCHKEY
  ERASE ERASEFLG STATUS STATFLG
MASTREC 40
```

Figure 490 Relative Record VSAM (RRDS) Random Update Without `OPTION VSAMCAT`

If you want to use the `VSAMCAT` feature, you must either include `OPTION VSAMCAT` in your program or set the `DYLISTL` macro keyword `VSAMCAT` to `Y`, and code `VSAM` in the `FILE` statement. If you include the `DD` or `DLBL` statement for the file in the JCL, then specifying the file type and its characteristics is optional.

For example:

```
FILE MSTRFL VSAM IO RANDOM SEARCHKEY
  ERASE ERASEFLG STATUS STATFLG
MASTREC 40
```

Figure 491 Relative Record VSAM (RRDS) Random Update with `OPTION VSAMCAT`

The following keywords are required:

<code>IO</code>	Allows the file to be updated in place. The record format and record size must be specified on the <code>FILE</code> statement in conjunction with this keyword, unless <code>VSAMCAT</code> is in effect and <code>VSAM</code> is coded in the <code>FILE</code> statement.								
<code>RANDOM</code> dataname	Any name not defined elsewhere in your program. Its length is always 4. It holds the relative record number of the record to be searched for.								
<code>STATUS</code> dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a <code>READ</code> has been issued for this file: <table> <tr> <td>blank</td><td>No record found.</td></tr> <tr> <td>Y</td><td>Record found.</td></tr> <tr> <td>E</td><td>End of file.</td></tr> <tr> <td>I</td><td>Duplicate or invalid key.</td></tr> </table>	blank	No record found.	Y	Record found.	E	End of file.	I	Duplicate or invalid key.
blank	No record found.								
Y	Record found.								
E	End of file.								
I	Duplicate or invalid key.								

The following keyword is optional:

<code>ERASE</code> dataname	Any name not defined elsewhere in your program. It is a 1-byte field that should contain an <code>E</code> to cause a <code>WRITE</code> command to delete the record just read.

Suppose that an RRDS VSAM file uses employee number as its relative record number and that a transaction file with records in no particular sequence has the same format. If the transaction employee number (EMPLOYEE) is not on the VSAM file, the employee is added. If the employee transaction matches a record on the VSAM file, replace the VSAM record or delete it (if the employee is terminated).

The following procedure logic can be coded:

```
FILE MSTRFL RRDS IO F 40
  RANDOM SEARCHKEY
  ERASE ERASEFLG
  STATUS STATFLG
MASTREC 40

FILE TRANS F 40 STATUS TRANSEOF
TRANREC 40
EMPLOYEE 5 1
TERMINATE 1 6
:
READTRAN:
  READ TRANS
  IF TRANSEOF EQ 'E' STOP ENDIF
  MOVE EMPLOYEE TO SEARCHKEY
  READ MSTRFL
  IF STATFLG NE 'Y' GOTO ADDRDN ENDIF
  IF TERMINATE EQ 'Y' MOVE 'E' TO ERASEFLG
  GOTO REWRITE ENDIF
  MOVE TRANREC TO MASTREC
REWRITE:
  WRITE MSTRFL
  GOTO READTRAN
ADDRDN:
  MOVE TRANREC TO MASTREC
  WRITE MSTRFL
  GOTO READTRAN
```

Figure 492 Code Procedure Logic Example (RRDS)

When the TRANS file reaches the end, the program issues a STOP command. You must specify when to end the run if an IO RANDOM update is used.

Before a record can be replaced, it must be read and found.

The file name MSTRFL must be the same as the file name or ddname specified in your JCL ([Figure 492](#)).

Sequential Insert and Extend

This section describes inserting new records into an existing KSDS or RRDS VSAM file, and writing additional records to the end of any VSAM file.

To extend a file, you need to describe the file to the operating system using JCL.

```
(VSE)      // ASSGN SYSnnn,X'cuu'
           // DLBL MYFILE,'MY.FILE.IDENT', ...
           // EXTENT SYSnnn, ...

(z/OS) //MYFILE DD DSN=MY.FILE.IDENT,DISP= ...
```

Figure 493 Extending a File Using JCL

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

For the statements used to extend VSAM files, see [Key Sequence VSAM \(KSDS\) Sequential Insert/Extend on page 521](#), [Entry Sequence VSAM \(ESDS\) Sequential Insert/Extend on page 522](#), and [Relative Record VSAM \(RRDS\) Sequential Insert/Extend on page 523](#).

Key Sequence VSAM (KSDS) Sequential Insert/Extend

Use VISION:Results statements to define the input file and the file to be extended (output).

If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code one of the following:

F	Fixed-length records.
V	Variable-length records.
S	Spanned variable-length records.

VSAM records are never blocked.

```
FILE TAPEIN FB 40 4800
FILE MYFILE KSDS OUTPUT FROM TAPEIN F STATUS OUTSTAT
```

Figure 494 Key Sequence VSAM (KSDS) Sequential Insert or Extend Without OPTION VSAMCAT

If you want to use the VSAMCAT feature, you must either include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

For example:

```
FILE TAPEIN
FILE MYFILE VSAM OUTPUT FROM TAPEIN STATUS OUTSTAT
```

Figure 495 Key Sequence VSAM (KSDS) Sequential Insert or Extend with OPTION VSAMCAT

To create variable-length records, see [Considerations for Processing Variable-Length Records on page 524](#).

The FILE statements above read each record on the input file and add it to the output file, unchanged. For information about selecting records, reformatting, or defining fields (data items), see *Advantage VISION:Results for z/OS Getting Started*. For record selection and data manipulation information, see [Chapter 7: Using the IF Command](#) and [Chapter 10: Using the MOVE Command](#).

The records on the input file must be in sequence by key for this run. In other words, each record written to the output file must have a key higher than the key in the previous record. Duplicates are not allowed.

REUSE must not be specified, so that the new input records are inserted among the existing records according to key. If the key of a new record matches that of an existing one, the write is ignored, the new record is dropped, and an I is returned in the STATUS field. A successful write causes a Y to be returned in the STATUS field.

The file name MYFILE must be the same as the file name or ddname specified in your JCL (above).

Entry Sequence VSAM (ESDS) Sequential Insert/Extend

Use VISION:Results statements to define the input file and the file to be extended (output).

If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code one of the following:

- F Fixed-length records.
- V Variable-length records.
- S Spanned variable-length records.

VSAM records are never blocked.

```
FILE TAPEIN FB 40 4800
FILE MYFILE ESDS OUTPUT FROM TAPEIN F
```

Figure 496 Entry Sequence VSAM (ESDS) Sequential Extend Without OPTION VSAMCAT

If you want to use the VSAMCAT feature, you must either include OPTION VSAMCAT in your program or set the DYLINSTL macro keyword VSAMCAT to Y, and code VSAM in the FILE statement. If you include the DD or DLBL statement for the file in the JCL, then specifying the file type and its characteristics is optional.

```
FILE TAPEIN
FILE MYFILE VSAM OUTPUT FROM TAPEIN
```

Figure 497 Entry Sequence VSAM (ESDS) Sequential Extend with OPTION VSAMCAT

To create variable-length records, see [Considerations for Processing Variable-Length Records on page 524](#).

The FILE statements above read each record on the input file and write it to the end of the output file, unchanged. For information about selecting records, reformatting, or defining fields (data items), see *Advantage VISION:Results for z/OS Getting Started*. For record selection and data manipulation information, see [Chapter 7: Using the IF Command](#) and [Chapter 10: Using the MOVE Command](#).

REUSE must not be specified when extending an ESDS file.

The file name MYFILE must be the same as the file name or ddname specified in the previous JCL.

As each record is added to an ESDS VSAM file, its relative byte address (RBA) is returned to you as either a 4-byte binary or 16-byte packed number. This number can be used as the record key for subsequent random access to the file. You *must* use RELBYTEX if there is any chance that the returned RBA will exceed 4GB in value. If you need this value, code:

```
FILE filename ESDS ... RELBYTE dataname
                        or
FILE filename ESDS ... RELBYTEX dataname
```

Figure 498 Coding in the FILE Statement Example 2

- Dataname contains the RBA of the record written as a 4-byte binary number when RELBYTE is used, or a 16-byte packed number when RELBYTEX is used.
- Dataname can be any data name that is not defined elsewhere in your program.

Relative Record VSAM (RRDS) Sequential Insert/Extend

Use VISION:Results statements to define the input file and the file to be extended (output). You also need one or more procedural statements to set up the relative record number (or slot number) of each output record.

If you have not specified OPTION VSAMCAT in your program or the keyword VSAMCAT=Y in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file, you must code F (for fixed-length records). VSAM records are never blocked.

```
FILE TAPEIN FB 40 4800
  BIN NUMBER 4 17 PD
FILE MYFILE RRDS OUTPUT FROM TAPEIN F RELREC BINNUM
  STATUS OUTSTAT
MOVE BIN_NUMBER TO BINNUM
```

Figure 499 Relative Record VSAM (RRDS) Sequential Insert or Extend Without OPTION VSAMCAT

If you want to use the VSAMCAT feature, you must either include `OPTION VSAMCAT` in your program or set the `DYLNSTL` macro keyword `VSAMCAT` to `Y`, and code `VSAM` in the `FILE` statement. If you include the `DD` or `DLBL` statement for the file in the `JCL`, then specifying the file type and its characteristics is optional.

```
FILE TAPEIN FB 40 4800
  BIN NUMBER 4 17 PD
FILE MYFILE VSAM OUTPUT FROM TAPEIN RELREC BINNUM
  STATUS OUTSTAT
MOVE BIN_NUMBER TO BINNUM
```

Figure 500 Relative Record VSAM (RRDS) Sequential Insert or Extend with `OPTION VSAMCAT`

The `RELREC` keyword is required. It must be followed by a data name not defined elsewhere in the program. The value of the data name becomes the relative record number of any given output record. Each record written to the file must have a relative record number higher than the previous record. Duplicates are not allowed.

The example above reads each record on the input file and adds it to the output file, unchanged, using the bin number value in each input record to establish the relative record number of each output record. For information about selecting records, reformatting, or defining fields (data items), see *Advantage VISION:Results for z/OS Getting Started*. For record selection and data manipulation information, see [Chapter 7: Using the IF Command](#) and [Chapter 10: Using the MOVE Command](#).

`REUSE` must not be specified, so that the new input records are inserted among the existing records according to relative record number. If the key of a new record matches that of an existing one, the write is ignored, the new record is dropped, and an `I` is returned in the `STATUS` field. A successful write causes a `Y` to be returned in the `STATUS` field.

The file name `MYFILE` must be the same as the file name or `ddname` specified in the previous `JCL`.

Considerations for Processing Variable-Length Records

`VISION:Results` simplifies the processing of files with variable-length records and spanned variable-length records. The processing for spanned variable-length records is the same as for variable-length records. A spanned record is larger than one VSAM control interval and therefore spans multiple VSAM control intervals.

Record Size

When specifying the logical record length for a variable file, you must specify the maximum data length.

Block Size

Blocking has no meaning for VSAM files and should be ignored.

Location of Data Within Record

VISION:Results considers position 1 of every variable-length VSAM record to be the first byte of your data, so you should start data definitions at that position. When the file name of a variable-length VSAM file is used as a data name, byte position 1 is referenced as the first byte of the record (the RDW).

Length of Current Record

On input, VISION:Results keeps track of the length of the data in the record just read. You can access this information by specifying the keyword LENGTH. A data name specified in this way is a 2-byte binary field and contains the length of the data in the record just read.

For output files, you must specify the length of the data portion of the current record.

Output from Itself

If a file is defined to be output from itself, it has its own IO area assigned. It also has a 2-byte binary length field assigned to the file. You can reference this length field by specifying the keyword LENGTH followed by a data name. It is your responsibility to move data to the area and place the length of the data in the LENGTH dataname before the record is to be written, either using the automatic cycle or a WRITE command. Failure to move a valid length to the LENGTH dataname can result in records of the wrong length being written.

Output from an Input file

When a file is written from an input file, VISION:Results does not assign an IO area for the file or a length field. VISION:Results references the IO area and length field for the input file when the output file is written. VISION:Results places restrictions on the type of file a variable output file can be written from. A variable VSAM file cannot be written from a variable non-VSAM file. If another combination is required, you must specify OUTPUT from itself for the file, move the data, and set up the data length.

When a record is being written from an input area, the length is automatically set up by VISION:Results when the record is read. LENGTH cannot be specified for the output file because the length field does not exist for this file. You can specify the LENGTH dataname for the input file if you want to adjust the length of the record. You can shorten a record to be written by changing the contents of the LENGTH dataname. Do not attempt to write a longer record.

If it is necessary to increase the length of a record, specify OUTPUT from itself on the FILE statement, move the data, and set up the length. The LENGTH dataname always contains the length of the data only.

File Printing of Variable Files

VISION:Results automatically handles the RDW.

Sorting a Variable File

To eliminate a possible error during SORT, variable-length records are moved from the IO area to a fixed-length area for processing. This move is transparent to you.

If an error occurs, the record shown in the IO area in the PRINTERROR dump is not necessarily the record you are processing. The variable record has been moved to a work area. Check your record there.

Numeric and Packed Key Considerations

When processing files where you are building records or retrieving records in random or skip mode, consider the sign of the key because the key has a data type of numeric (zoned decimal) or packed. A positive sign for a numeric or packed field can be C or F. This can be a problem if you are searching for a particular record on a VSAM file and the key you are using to search has a different sign than the key in the record. For example, a 2-byte packed field with the value 123C is not considered equal to a 2-byte packed field with the value 123F. A NO RECORD FOUND status is set.

In a VISION:Results program, you can encounter an unlike sign problem under the following circumstance: you have set up a packed key in the VSAM key area and this key has developed a C sign (perhaps the key had been converted from numeric to packed and a C sign resulted from this process). If the key in the record contains an F sign, a status code of blank (no record found) is returned to the program after a random retrieval read. The same situation can occur with numeric fields.

To prevent or correct for unlike signs, treat the keys as character data. For example:

```
FILE VSAMIN KSDS F 120 RANDOM SEARCHKEY
  STATUS STATUSFLAG KEYLEN 3
FILE TRANIN STATUS TRANEOF
  KEYIN 3 CH ; (key is actually packed decimal)
PROCEDURE:
  IF TRANEOF EQ 'E' STOP ENDIF
  MOVE KEYIN TO SEARCHKEY ; (sign remains unchanged)
  READ VSAMIN
  IF STATUSFLAG NE 'Y' GOTO NOTFOUND ENDIF
  .
  .
  .
```

Figure 501 Numeric and Packed Key Considerations

Note: Throughout this section, items shown in **bold** are optional when using the OPTION VSAMCAT statement or the VSAMCAT=Y keyword in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file. When using the VSAMCAT feature, you must enter VSAM in the FILE statement. Otherwise, the VSAMCAT feature is not used for that file.

In [Figure 501](#), the keys actually have a data type of packed but are defined as character, so no sign change can occur. If the key has to be treated as packed for other purposes, it can be redefined.

```
FILE VSAMIN2 KSDS F 130 RANDOM FINDKEY
  STATUS VSSTAT KEYLEN 5
ACCOUNT 5 NU ...
FILE TRANSIN STATUS TRANSTAT
  TRANACCT 5 ...
PROCEDURE:
  IF TRANSTAT EQ 'E' STOP ENDIF
  MOVE TRANACCT TO FINDKEY
  READ VSAMIN2
  .
  .
  .
```

Figure 502 Numeric and Packed Key Considerations Where Key is Treated as Packed

In [Figure 502](#), a numeric key in the transaction file (TRANSIN) is defined as character. FINDKEY is also defined as character (default). No sign change happens when TRANACCT is moved to FINDKEY.

To convert a packed key's sign from C to F, you can OR the field. For example:

```
COMBINE BITS X'000F' OR SRCHKEY
```

Figure 503 Convert a Packed Key's Sign From C to F

In [Figure 503](#), a 2-byte packed key is having its sign set to F.

To convert a packed key's sign from F to C, you can AND the field. For example:

```
COMBINE BITS X'FFFC' AND SRCHKEY
```

Figure 504 Convert a Packed Key's Sign From F to C

In [Figure 504](#), a 2-byte packed key with an F sign is having its sign set to C.

KSDS Examples

Note: Throughout this section, items shown in **bold** are optional when using the OPTION VSAMCAT statement or the VSAMCAT=Y keyword in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file. When using the VSAMCAT feature, you must enter VSAM in the FILE statement. Otherwise, the VSAMCAT feature is not used for that file.

Example 1 Fixed-Length KSDS In, KSDS Out

```
FILE KSDSIN KSDS F 140  
FILE KSDSOUT KSDS OUTPUT FROM KSDSIN F 140 REUSE
```

Figure 505 [Example 1](#) Fixed-Length KSDS In, KSDS Out

The entire input KSDS file is copied to another KSDS file.

The REUSE option must be specified when the output file was defined or altered using IDCAMS.

Any data already in the KSDSOUT file is replaced with the contents of the input file.

Example 2 Variable-Length KSDS In, Variable-Length File Out

```
FILE KSDSIN KSDS V 120 LENGTH INLEN  
FILE TAPEOUT SYS006 TAPE OUTPUT FROM TAPEOUT  
  VB 120 2484 LENGTH OUTLEN  
MOVE KSDSIN TO TAPEOUT LENGTH INLEN  
MOVE INLEN TO OUTLEN
```

Figure 506 [Example 2](#) Variable-Length KSDS In, Variable-Length File Out

The entire KSDS file is backed up to tape.

Because the output file is defined in the FILE statement as having variable-length records, a standard record descriptor word (LLbb) is built on the front of each output record before it is written.

The data length of the output record must be moved to the output file LENGTH dataname on each cycle.

Example 3 Variable-Length File In, KSDS Out

```
FILE TAPEIN TAPE SYS029 VB 220 4484 LENGTH INLEN
FILE KSDSOUT KSDS OUTPUT FROM KSDSOUT
  V 220 LENGTH OUTLEN REUSE
MOVE TAPEIN TO KSDSOUT LENGTH INLEN
MOVE INLEN TO OUTLEN
```

Figure 507 [Example 3](#) Variable-Length File In, KSDS Out

The entire input file is copied to a KSDS file.

Because the input tape FILE statement defines the records as variable, VISION:Results removes the record descriptor (LLbb) and moves the length of the data to the input LENGTH dataname.

The data length of the output record must be moved to the output file LENGTH dataname on each cycle.

The REUSE option must be specified when the KSDS file is defined or altered using IDCAMS.

Any data already in the KSDSOUT file is replaced with the contents of the input file.

Example 4 Fixed-Length File In, KSDS Out, with SORT

```
FILE TAPEIN SYS041 FB 95 2850 TAPE
  CUSTNUM 9 2
FILE KSDSOUT KSDS OUTPUT FROM TAPEIN F 95 REUSE
SORT TAPEIN USING CUSTNUM WORK 3
```

Figure 508 [Example 4](#) Fixed-Length File In, KSDS Out, with SORT

The entire input file is sorted by key. The sorted records are then written to a KSDS file.

No JCL is supplied for SORTIN or SORTOUT; however, all other JCL normally required for a sort is needed.

The REUSE option must be specified when the KSDS file is defined or altered using IDCAMS.

Any data already in the KSDSOUT file is replaced with the sorted records.

Example 5 Skip Sequential Read Fixed-Length KSDS

```

FILE INVMAST KSDS SKIP PRODCODE STATUS SRCHFLAG F 75
KEYLEN 8 PARTKEY 2
MSTCLASS 2 3 (CLASS)
MSTPRODUCT 6 (PRODUCT CODE)
QTYONHAND 5 29 PD Z (ON HAND)
QTYORDRD 5 42 PD Z (QUANTITY ORDERED)
ORDRDATE 6 47 NU D (DATE ORDERED)

FILE FINDERS SYS007 FB 80 1600 STATUS ENDFLAG
CLASSCODE 2

IF ENDFLAG EQ 'E' STOP ENDIF
MOVE CLASSCODE TO PRODCODE LENGTH 2
READINV:
READ INVMAST
IF SRCHFLAG EQ 'E' REJECT ENDIF
IF MSTCLASS GT CLASSCODE MOVE 66 TO DYLLINE
REJECT ENDIF
LIST SUPPRESS MSTCLASS MSTPRODUCT ORDRDATE QTYORDRD QTYONHAND
GOTO READINV

T1 'INVENTORY REPORT FOR SELECTED PRODUCT CLASSES' WITH 2 AFTER
T1+120 DYLLDATE

```

Figure 509 Example 5 Skip Sequential Read Fixed-Length KSDS

A generic search by class code is performed.

PARTKEY defines the length of the partial (generic) key.

All records found for each class on the FINDERS file are written to the report. Each class begins on a new page.

Example 6 Random Read Fixed-Length KSDS

```

FILE SALES TAPE SYS035 FB 128 1280 STATUS ALLDONE
SALESMAN 3 3
CUSTNUM 6 26
PRODUCT 7 32
QUANTITY 3 54 PD Z
UNITPRICE 4 81 PD 2 E

FILE CUSMSTR KSDS RANDOM FINDCUST F 225 KEYLEN 6 STATUS MSTRSTAT
CUSTOMER 25 8

WORKAREA
CNAME 25 (CUSTOMER NAME)
TOT 7 PD 2 E
TALLY E 3

IF ALLDONE EQ 'E' STOP ENDIF
MOVE CUSTNUM TO FINDCUST
READ CUSMSTR
IF MSTRSTAT EQ 'Y' MOVE CUSTOMER TO CNAME
ELSE MOVE '*** NOT ON FILE ***' TO CNAME ENDIF
TOT = QUANTITY * UNITPRICE
LIST SUPPRESS SALESMAN CNAME PRODUCT UNITPRICE QUANTITY TOT
CONTROL SALESMAN
ON CHANGE IN SALESMAN
LIST 'TOTAL SALESMAN' AT CNAME TALLY SUM TOT
WITH 2 BEFORE AND EJECT AFTER
ON FINAL
LIST '*** FINAL TOTAL ***' AT PRODUCT SUM TOT
T1 'S A L E S B Y S A L E S M A N' WITH 2 AFTER
T1+100 DYLLDATEPAG

```

Figure 510 [Example 6](#) Random Read Fixed-Length KSDS

A random search of a KSDS file (CUSMSTR) is performed to obtain the customer's name for a sales report line.

Example 7 Random Update Variable-Length KSDS

```

FILE ACCTMST KSDS IO RANDOM ACCTKEY STATUS MSTFLAG V 200
  LENGTH MSTLEN KEYLEN 6 ERASE KILLIT MASTREC 200

FILE TRANS DISK 3350 VB 200 2044 STATUS TRANSEOF LENGTH TRANSLN
TRANREC 200
TRANCODE 1 1
TRANACCT 6 5

READTRAN:
  READ TRANS
  IF TRANSEOF EQ 'E' STOP ENDIF
  MOVE TRANACCT TO ACCTKEY
  READ ACCTMST
  IF TRANCODE EQ 'A' GOTO ADDRTRN ENDIF
  IF TRANCODE EQ 'D' GOTO DELRTN ENDIF
CHGRTN:
  IF MSTFLAG NE 'Y' MOVE 'BAD CHANGE' TO DYLPRTCOMM
  PRINT TRANREC GOTO READTRAN ENDIF
  MOVE TRANREC TO MASTREC LENGTH TRANSLN
  MOVE TRANSLN TO MSTLEN
  GOTO WRITEMST
ADDRTRN:
  IF MSTFLAG EQ 'Y' MOVE 'BAD ADDN' TO DYLPRTCOMM
  PRINT TRANREC PRINT MASTREC GOTO READTRAN ENDIF
  MOVE TRANREC TO MASTREC LENGTH TRANSLN
  MOVE TRANSLN TO MSTLEN
  GOTO WRITEMST
DELRTN:
  IF MSTFLAG NE 'Y' MOVE 'BAD DELETE' TO DYLPRTCOMM
  PRINT TRANREC GOTO READTRAN ENDIF
  MOVE 'E' TO KILLIT
WRITEMST:
  WRITE ACCTMST
  GOTO READTRAN

```

Figure 511 [Example 7](#) Random Update Variable-Length KSDS

A KSDS file is randomly updated by adding new records, replacing existing records, and deleting (erasing) others.

The length of an existing record in a KSDS file can be changed by altering the LENGTH prior to rewriting. (See CHGRTN:).

ESDS Examples

Note: Throughout this section, items shown in **bold** are optional when using the OPTION VSAMCAT statement or the VSAMCAT=Y keyword in the DYLINSTL macro and the JCL does not include the DD or DLBL statement for the file. When using the VSAMCAT feature, you must enter VSAM in the FILE statement. Otherwise, the VSAMCAT feature is not used for that file.

Example 1 Fixed-Length ESDS In, ESDS Out

```
FILE ESDSIN ESDS F 140  
FILE ESDSOUT ESDS OUTPUT FROM ESDSIN F 140 REUSE
```

Figure 512 [Example 1](#) Fixed-Length ESDS In, ESDS Out

The entire input ESDS file is copied to another ESDS file.

The REUSE option must be specified when the output file was defined or altered using IDCAMS.

Any data already in the ESDSOUT file is replaced with the contents of the input file.

Example 2 Variable-Length ESDS In, Variable-Length File Out

```
FILE ESDSIN ESDS V 120 LENGTH INLEN  
FILE TAPEOUT TAPE OUTPUT FROM TAPEOUT  
  VB 120 2484 LENGTH OUTLEN SYS006  
MOVE ESDSIN TO TAPEOUT LENGTH INLEN  
MOVE INLEN TO OUTLEN
```

Figure 513 [Example 2](#) Variable-Length ESDS In, Variable-Length File Out

The entire ESDS file is backed up to tape.

Because the output file is defined in the FILE statement as having variable-length records, a standard record descriptor word (LLbb) is built by VISION:Results on the front of each output record before it is written.

The data length of the output record must be moved to the output file LENGTH dataname on each cycle.

Example 3 Variable-Length File In, ESDS Out

```
FILE TAPEIN TAPE SYS029 VB 220 4484 LENGTH INLEN  
FILE ESDSOUT ESDS OUTPUT FROM ESDSOUT V 220 LENGTH OUTLEN REUSE  
MOVE TAPEIN TO ESDSOUT LENGTH INLEN  
MOVE INLEN TO OUTLEN
```

Figure 514 [Example 3](#) Variable-Length File In, ESDS Out

The entire input file is copied to an ESDS file.

Because the input tape FILE statement defines the records as variable, VISION:Results removes the record descriptor (LLbb) and moves the length of the data to the input LENGTH dataname.

The data length of the output record must be moved to the output file LENGTH dataname on each cycle.

The REUSE option must be specified when the ESDS file is defined or altered using IDCAMS.

Any data already in the ESDSOUT file is replaced with the contents of the input file.

Example 4 Fixed-Length File In, ESDS Out, with SORT

```
FILE TAPEIN SYS041 FB 95 2850 TAPE
  CUSTNUM 9 2
FILE ESDSOUT ESDS OUTPUT FROM TAPEIN F 95 REUSE
SORT TAPEIN USING CUSTNUM WORK 3
```

Figure 515 [Example 4](#) Fixed-Length File In, ESDS Out, with SORT

The entire input file is sorted by key. The sorted records are then written to an ESDS file.

The REUSE option must be specified when the ESDS file is defined or altered using IDCAMS.

Any data already in the ESDSOUT file is replaced with the sorted records.

Example 5 Variable-Length File In, ESDS Out (ESDSOUT), RBAs Saved in a Sequential Output File

```
FILE BACKUP SYS015 TAPE VB 200 2044 LENGTH INLGTH
  CUSTOMER 8 5
FILE ESDSOUT ESDS OUTPUT FROM ESDSOUT RELBYTE RBA V 200
  LENGTH OUTLGTH REUSE
FILE CUSTKEY SYS016 TAPE FB 12 1200 OUTPUT FROM CUSTKEY
  CUSTOUT 8
  RBAOUT 4 BI

MOVE BACKUP TO ESDSOUT LENGTH INLGTH
MOVE INLGTH TO OUTLGTH
WRITE ESDSOUT
MOVE CUSTOMER TO CUSTOUT
MOVE RBA TO RBAOUT
```

Figure 516 [Example 5](#) Variable-Length File In, ESDS Out, RBAs Saved in a Sequential Output File

A backup file is used to restore a variable-length ESDS file. After each VSAM record is written, its customer number and Relative Byte Address are saved in the output area for the CUSTKEY file. The resulting file can be used at a later time to find a given customer, directly, on the ESDS file. See [Random Read on page 501](#) and [Random Update on page 513](#).

The REUSE option must be specified when the ESDS file is defined or altered using IDCAMS.

Any data already in the ESDSOUT file is replaced with the contents of the input file.

Chapter 21: ISAM Processing

Under the indexed sequential access method or [ISAM](#) file organization, VISION:Results supports fixed and, for z/OS users, variable-length records.

The input and/or output modes supported are sequential retrieval, sequential load/load extend/insert, random retrieval, random retrieval and update, and limited sequential (skip) retrieval by full or generic key.

ISAM

Note: IBM withdrew support for ISAM beginning with z/OS r1.7. Therefore, VISION:Results cannot support access to ISAM data sets under operating systems after z/OS r1.6.

Syntax Format

The keywords and operands pertaining only to [ISAM](#) are described in this chapter. Operands such as filename, recordsize, and dataname are described in detail in [Chapter 5: Data Name Qualification](#).

```
FILE filename
ISAM
{INPUT | OUTPUT FROM filename | IO}
[{FB | VB | F | V} [recordsize [blocksize]]]
{RANDOM dataname2 {CH | NU | BI | PD} [decimals] |
  SKIP dataname2 {CH | NU | BI | PD} [decimals] | SEQUENTIAL}
[KEYLEN nnn [KEYLOC nnnn]] [POSITION dataname2]
[STATUS dataname2] [COUNT dataname2] [LENGTH dataname2]
[BYPASS nn] [DROPER] [DUMMY] [NULL]
[RETAIN | NORETAIN]
[[MODIFY modulename [nK] [SIZE nnnnn]
  [PARM ({IOAREA | WORK | dataname3...})]] [DYNAM]
{DISK {2311 | 2314 | 3330 | 3340 }}]
[EXTENTS nn] [MASTER] [CYLOFL nn]
[SYSnnn] [ONEBUFF]
```

Figure 517 ISAM File Syntax Format

Creating a File

This section describes the creation of a new [ISAM](#) file or the replacement of an old one. If you intend to read or update existing files only, you do not need to use this procedure.

To create a file, describe the file to the operating system using JCL. For example:

```
(VSE) // ASSGN SYSnnn,X'cuu'
      // DLBL MYFILE,'MY.FILE.IDENT', ...
      // EXTENT SYSnnn, ...

(z/OS) //MYFILE DD DSN=MY.FILE.IDENT,DISP= ...
```

Figure 518 Create an ISAM File Using JCL

The particulars of the JCL required vary from one installation to another. To successfully code the JCL, you must be familiar with your installation standards.

ISAM Create

Use VISION:Results statements to define the input file and the file to be created (output):

```
(VSE)  FILE TAPEIN FB 40 4800
       FILE MYFILE ISAM OUTPUT FROM TAPEIN DISK 3340
       FB 40 1600 KEYLEN 5 KEYLOC 2
       CYLOFL 3 EXTENTS 2  (optional parameters)

(z/OS) FILE TAPEIN FB 40 4800
       FILE MYFILE ISAM OUTPUT FROM TAPEIN
       FB 40 1600 KEYLEN 5 KEYLOC 2
```

Figure 519 Defining the Input File and the Output File

KEYLEN nnn	Specifies the length of the record key. The allowable values of nnn are 1 to 255. This is a required parameter for VSE systems.
KEYLOC nnnn	Specifies the starting location of the key within the data portion of the record. The allowable values of nnnn are 1 to 4095. This is a required parameter for VSE systems.

To create variable-length records, see [Considerations for Processing Variable-Length Records on page 556](#).

The records on the input file must be in sequence by key for this run. Each record written to the output file must have a key higher than the key in the previous record. Duplicates are not allowed.

The previous example reads each record on the input file and copies it to the output file, unchanged. If record selection or reformatting is required and you need information about defining fields (data items), see *Advantage VISION:Results for z/OS Getting Started*. For information about record selection and data manipulation, see [Chapter 7: Using the IF Command](#) and [Chapter 10: Using the MOVE Command](#).

The file name MYFILE must be the same as the file name or ddname specified in the previous JCL.

z/OS

z/OS users can specify any or all of the second line of the [ISAM](#) FILE statement as DCB subparameters in their JCL.

Various options can be specified using the OPTCD subparameter of the DCB parameter in the JCL. For more information, see the *IBM JCL Reference Manual*.

VSE

The DISK parameter is required and is coded as follows:

DISK nnnn Defines the type of disk that contains the file. Allowable values are 2311, 2314, 3330, and 3340.

The following additional keywords are optional for VSE:

CYLOFL nn Where nn is the number of tracks to be reserved on each cylinder for cylinder overflow. The default is 0. Specify this parameter if you expect to add records to the file after it has been initially loaded. An independent overflow area, also used to hold inserted records, can be used in addition to or instead of cylinder overflow.

MASTER This keyword specifies that a master index is allocated for this file.

EXTENTS nn Where nn is the total number of extents allocated to the file. The default is 3 (cylinder index, prime data, and independent overflow). The value for nn must be between 2 and 20.

Each extent is identified in the JCL by an extent type code:

master index	4
cylinder index	4
prime data	1
independent overflow	2

Each extent is further numbered from 1 up (unless there is a master index that has an extent sequence number of 0).

Typical JCL for an [ISAM](#) file create run is:

```
// ASSGN SYS005,X'190'  
// ASSGN SYS006,X'191'  
// DLBL MYFILE,'MY.FILE.IDENT',90/365,ISC  
// EXTENT SYS005,111111,4,1,...Cylinder index  
// EXTENT SYS005,111111,1,2,...Prime data  
// EXTENT SYS006,123456,2,3,...Independent overflow
```

Figure 520 JCL for an ISAM File Create Run

The ISC subparameter on the previous DLBL statement must be specified when creating an [ISAM](#) file.

VSE [ISAM](#) files having one record to a block (unblocked) require special consideration and should be avoided. If you must create this type of file, see [Unblocked ISAM Files in VSE Considerations on page 558](#).

Sequential Read

This section describes reading an [ISAM](#) file starting with the first record and then reading serially until end of file occurs or a STOP command is issued by the VISION:Results program.

To read a file, describe the file to the operating system using JCL.

```
(VSE) // ASSGN SYSnnn,X'cuu'
      // DLBL MYFILE,'MY.FILE.IDENT', ...
      // EXTENT SYSnnn, ...

(z/OS) //MYFILE DD DSN=MY.FILE.IDENT,DISP=SHR
```

Figure 521 Reading a File Using JCL

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

ISAM Sequential Read

Use a FILE statement to define the input file:

```
(VSE) FILE MYFILE ISAM DISK 3340
      FB 40 1600 KEYLEN 5 KEYLOC 2
      CYLOFL 3 EXTENTS 2 (optional parameters)

(z/OS) FILE MYFILE ISAM FB 40
```

Figure 522 ISAM Sequential Read

To read variable-length records, see [Considerations for Processing Variable-Length Records on page 556](#).

The previous example reads each record on the input file in ascending order by key and makes the record available to any procedure logic that follows. For information about defining fields (data items), see *Advantage VISION:Results for z/OS Getting Started*. For information about record selection and data manipulation, see [Chapter 7: Using the IF Command](#) and [Chapter 10: Using the MOVE Command](#).

The file name MYFILE must be the same as the file name or ddname specified in the previous JCL.

VSE Considerations

DISK, KEYLEN, and KEYLOC are required parameters.

DISK nnnn	Defines the type of disk that contains the file. The allowable values are 2311, 2314, 3330, 3340, and 3350.
KEYLEN nnn	Specifies the length of the record key. The allowable values of nnn are 1 to 255. This is a required parameter for VSE systems.
KEYLOC nnnn	Specifies the starting location of the key within the data portion of the record. The allowable values of nnnn are 1 to 4095. This is a required parameter for VSE systems.

The following additional keywords are optional for VSE:

CYLOFL nn	Where nn is the number of tracks reserved on each cylinder for cylinder overflow. The default is 0. This parameter must be supplied if cylinder overflow was specified when the file was initially loaded.								
MASTER	This keyword specifies that a master index was allocated for this file when it was loaded.								
EXTENTS nn	Where nn is the total number of extents allocated to the file. The default is 3 (cylinder index, prime data, and independent overflow). The value for nn must be between 2 and 20. Each extent is identified in the JCL by an extent type code: <table><tr><td>master index</td><td>4</td></tr><tr><td>cylinder index</td><td>4</td></tr><tr><td>prime data</td><td>1</td></tr><tr><td>independent overflow</td><td>2</td></tr></table> Each extent is further numbered from 1 up (unless there is a master index that has an extent sequence number of 0).	master index	4	cylinder index	4	prime data	1	independent overflow	2
master index	4								
cylinder index	4								
prime data	1								
independent overflow	2								

Typical JCL for reading an [ISAM](#) file is:

```
// ASSGN SYS005,X'190'  
// ASSGN SYS006,X'191'  
// DLBL MYFILE,'MY.FILE.IDENT',90/365,ISE  
// EXTENT SYS005,111111,4,1,... Cylinder index  
// EXTENT SYS005,111111,1,2,... Prime data  
// EXTENT SYS006,123456,2,3,... Independent overflow
```

Figure 523 Typical JCL for Reading an ISAM File

The ISE subparameter on the DLBL statement (above) must be specified when reading an [ISAM](#) file.

VSE [ISAM](#) files having one record to a block (unblocked) require special consideration. If you must read this type of file, see [Unblocked ISAM Files in VSE Considerations on page 558](#).

Skip Sequential Read

This section describes reading an [ISAM](#) file by skipping to a particular record in the file and reading records serially from that point until some condition is met or end of file is encountered. In either case, another similar retrieval can be initiated, and so on, as often as needed. You must use a READ command to read the file, and a STOP statement must be coded to end the run unless your program has a SORT.

To read either type of file, describe the file to the operating system using JCL.

```
(VSE) // ASSGN SYSnnn,X'cuu'
      // DLBL MYFILE,'MY.FILE.IDENT', ...
      // EXTENT SYSnnn, ...

(z/OS) //MYFILE DD DSN=MY.FILE.IDENT,DISP=SHR
```

Figure 524 Skip Sequential Read

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

ISAM Skip Sequential Read

Use a FILE statement to define the input file:

```
(VSE) FILE MYFILE ISAM SKIP SEARCHKEY STATUS STATFLG
      DISK 3340 FB 40 1600 KEYLEN 5 KEYLOC 2
      CYLOFL 3 EXTENTS 2 (optional parameters)
      POSITION POSCODE (optional)

(z/OS) FILE MYFILE ISAM SKIP SEARCHKEY STATUS STATFLG
      FB 40 KEYLEN 5
      POSITION POSCODE (optional)
```

Figure 525 ISAM Skip Sequential Read

The following keywords are required:

SKIP dataname	Any name not defined elsewhere in your program. Its length is equal to the key length. It holds the key of the record to be retrieved. A CH (character) data type is assumed. If your key is signed numeric, packed decimal, or binary, code NU, PD, or BI, respectively, after dataname: SKIP dataname PD Also, if the number of decimal places is other than zero, code it after the data type: SKIP dataname PD 2
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: blank No record found. Y Record found. E End of file. I Duplicate or invalid key.
KEYLEN nnn	Specifies the length of the record key. The allowable values for nnn are 1 to 255.

The following keyword is optional:

POSITION dataname	Any name not defined elsewhere in your program. It is a 1-byte field which can be used to control the type of search (positioning) as follows:
blank / G (default)	Start reading at the first record in which the key is equal to or greater than the value in the SKIP dataname.
E	Start reading at the record in which the key is equal to the value in the SKIP dataname.
R	Repeat the previous positioning.

VISION:Results recognizes the need to skip to a new starting point in the file by comparing the contents of the SKIP dataname with its previous value. To restart the process with the same SKIP key as the previous time (if required), move an R to the POSITION dataname.

Code procedure logic to position the file based on a starting key and read records until some defined condition is met:

```
.
NXTGROUP:
  READ INKEYS
  IF INKEYFLG EQ 'E' STOP ENDIF
  MOVE STARTKEY TO SEARCHKEY
  MOVE 'G' TO POSCODE (optional)
READLOOP:
  READ MYFILE
  IF STATFLG NE 'Y' GOTO NXTGROUP ENDIF
  IF ISAMKEY GT ENDKEY GOTO NXTGROUP ENDIF
  HEXPRINT ISAMREC
  GOTO READLOOP
```

Figure 526 Code Procedure Logic

In the previous example, each record in a file INKEYS contains a STARTKEY and an ENDKEY. The program initializes the SKIP dataname (SEARCHKEY) and reads the first record on the [ISAM](#) file whose key is equal to or greater than the starting key. It then checks to make sure that a record was found. Finally, the key of the record just read is compared to the highest key required (ENDKEY). All records on the file whose keys fall within the limits are printed, and the program returns to read the next INKEYS record.

When the INKEYS file reaches the end, the program issues a STOP command. You must always specify when to end the run if a skip sequential file is used.

To read variable-length records, see [Considerations for Processing Variable-Length Records on page 556](#).

The MYFILE filename must be the same as the file name or ddname specified in the previous JCL.

Generic Search

A generic search uses a partial key to point to the group of records required. Suppose that the first 3 bytes of your key contain a department number and you want to obtain all the records for Department 341. Just fill the SKIP dataname with binary zeros, and then move in the partial key:

```
WORKAREA
.
DEPT341 3 VALUE '341'
.
MOVE X'0000000000' TO SEARCHKEY
MOVE DEPT341 TO SEARCHKEY LENGTH 3
READLOOP:
  READ MYFILE etc.
  .
  GOTO READLOOP
```

Figure 527 Generic Search

This procedure starts reading records at the first one whose key is equal to or greater than 341, followed by 2 bytes of binary 00s.

VSE Considerations

DISK and KEYLOC are additional required parameters:

DISK nnnn	Defines the type of disk that contains the file. The allowable values are 2311, 2314, 3330, 3340, and 3350.
KEYLOC nnnn	Specifies the starting location of the key within the data portion of the record. The allowable values of nnnn are 1 to 4095.

The following additional keywords are optional for VSE:

CYLOFL nn	Where nn is the number of tracks reserved on each cylinder for cylinder overflow. The default is 0. This parameter must be supplied if cylinder overflow was specified when the file was initially loaded.

MASTER	This keyword specifies that a master index was allocated for this file when it was loaded.

EXTENTS nn	Where nn is the total number of extents allocated to the file. The default is 3 (cylinder index, prime data, and independent overflow). The value of nn must be between 2 and 20.

Each extent is identified in the JCL by an extent type code:

master index	4
cylinder index	4
prime data	1
independent overflow	2

Each extent is further numbered from 1 up (unless there is a master index that has an extent sequence number of 0).

Typical JCL for reading an [ISAM](#) file is:

```
// ASSGN SYS005,X'190'  
// ASSGN SYS006,X'191'  
// DLBL MYFILE,'MY.FILE.IDENT',90/365,ISE  
// EXTENT SYS005,111111,4,1,... Cylinder index  
// EXTENT SYS005,111111,1,2,... Prime data  
// EXTENT SYS006,123456,2,3,... Independent overflow
```

Figure 528 Typical JCL for Reading an ISAM File

The ISE subparameter on the DLBL statement must be specified when reading an [ISAM](#) file.

VSE [ISAM](#) files having one record to a block (unblocked) require special consideration. If you must read this type of file, see [Unblocked ISAM Files in VSE Considerations on page 558](#).

Random Read

This section describes reading records, one at a time and in no particular sequence, from an [ISAM](#) file. The program must identify precisely which record is wanted prior to issuing the READ request. After the READ, one of two conditions exists: either the requested record is found, or it is not found. If the record is not found, the program must not attempt to access the record area because it may not have been allocated.

To read a file, you need to describe the file to the operating system using JCL.

```
(VSE) // ASSGN SYSnnn,X'cuu'  
      // DLBL MYFILE,'MY.FILE.IDENT', ...  
      // EXTENT SYSnnn, ...  
  
(z/OS) //MYFILE DD DSN=MY.FILE.IDENT,DISP=SHR
```

Figure 529 Random Read

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

ISAM Random Read

Use a FILE statement to define the input file:

```
(VSE) FILE MYFILE ISAM RANDOM SEARCHKEY STATUS STATFLG  
      DISK 3340 FB 40 1600 KEYLEN 5 KEYLOC 2  
      CYLOFL 3 EXTENTS 2 (optional parameters)  
  
(z/OS) FILE MYFILE ISAM RANDOM SEARCHKEY STATUS STATFLG  
      FB 40 KEYLEN 5
```

Figure 530 ISAM Random Read

The following keywords are required:

RANDOM dataname	Any name not defined elsewhere in your program. Its length is equal to the key length. It holds the key of the record to be retrieved. A CH (character) data type is assumed. If your key is signed numeric, packed decimal, or binary, code NU, PD, or BI, respectively, after dataname: RANDOM dataname PD Also, if the number of decimal places is other than zero, code it after the data type: RANDOM dataname PD 2
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: blank No record found. Y Record found. E End of file. I Invalid or duplicate key.
KEYLEN nnn	Specifies the length of the record key. The allowable values for nnn are 1 to 255.

Code procedure logic to set up the search key, read, and check the result:

```

:
:
READLOOP:
  READ TAPEIN
  IF TAPESTAT EQ 'E' STOP ENDIF
  MOVE EMPLOYEE TO SEARCHKEY
  READ MYFILE
  IF STATFLG NE 'Y' GOTO NOTFOUND ENDIF
  HEXPRINT ISAMREC
  GOTO READLOOP
:
:

```

Figure 531 Code Procedure Logic

Above, each record in a file TAPEIN contains an employee number (EMPLOYEE). The program moves this number to the RANDOM dataname as a key and issues a READ for the [ISAM](#) file. If the record is not found, the program branches to an error routine. If the record is found, it is printed and the program returns to read another TAPEIN record.

When the TAPEIN file reaches the end, the program issues a STOP command. It is always necessary to tell VISION:Results when to end the run if a RANDOM read is used.

To read variable-length records, see [Considerations for Processing Variable-Length Records on page 556](#).

The MYFILE filename must be the same as the file name or ddname specified in the previous JCL.

VSE Considerations

DISK and KEYLOC are additional required parameters.

DISK nnnn Defines the type of disk that contains the file. The allowable values are 2311, 2314, 3330, 3340, and 3350.

KEYLOC nnnn Specifies the starting location of the key within the data portion of the record. The allowable values of nnnn are 1 to 4095.

The following additional keywords are optional for VSE:

CYLOFL nn Where nn is the number of tracks reserved on each cylinder for cylinder overflow. The default is 0. This parameter must be supplied if cylinder overflow was specified when the file was initially loaded.

MASTER This keyword specifies that a master index was allocated for this file when it was loaded.

EXTENTS nn Where nn is the total number of extents allocated to the file. The default is 3 (cylinder index, prime data, and independent overflow). The value of nn must be between 2 and 20.

Each extent is identified in the JCL by an extent type code:

master index	4
cylinder index	4
prime data	1
independent overflow	2

Each extent is further numbered from 1 up (unless there is a master index that has an extent sequence number of 0).

Typical JCL for reading an [ISAM](#) file is:

```
// ASSGN SYS005,X'190'  
// ASSGN SYS006,X'191'  
// DLBL MYFILE,'MY.FILE.IDENT',90/365,ISE  
// EXTENT SYS005,111111,4,1,... Cylinder index  
// EXTENT SYS005,111111,1,2,... Prime data  
// EXTENT SYS006,123456,2,3,... Independent overflow
```

Figure 532 Typical JCL for Reading an ISAM File

The ISE subparameter on the DLBL statement must be specified when reading an [ISAM](#) file.

VSE [ISAM](#) files having one record to a block (unblocked) require special consideration. If you must read this type of file, see [Unblocked ISAM Files in VSE Considerations on page 558](#).

Sequential Update

This section describes reading each record in an [ISAM](#) file, starting at the beginning, and updating the file in place (that is, without copying the old file to a new file). You must use a READ command to read the file, and a STOP command must be coded to end the run, unless your program has a SORT.

Note: This type of run is not restartable. After one record has been added, changed, or deleted, the file is not the same, regardless of why the job ended. As a precaution before running a program that uses this technique, create a method of recovering the original file.

You can add new records, change existing records, and flag records for deletion. Generally speaking, deleted records remain physically in the file until removed by a backup and restore (re-org) run.

To read a file, describe the file to the operating system using JCL.

```
(VSE)      // ASSGN SYSnnn,X'cuu'
           // DLBL MSTRFL,'MY.FILE.IDENT',    ...
           // EXTENT SYSnnn,    ...

(z/OS) //MSTRFL DD DSN=MY.FILE.IDENT,DISP=OLD
```

Figure 533 Sequential Update

Note the disposition of OLD. This choice (also available under VSE, Levels 2 and up) prevents other users from accessing the file while your program is updating it.

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

ISAM Sequential Update

Two file definitions of the [ISAM](#) file are required. Use VISION:Results statements like these:

```
(VSE)      FILE MSTRFL ISAM STATUS MASTFLAG
           DISK 3340 FB 40 1600 KEYLEN 5 KEYLOC 2
           CYLOFL 3 EXTENTS 2 (optional parameters)
           MASTREC 40 MASTEMPL 5 2

           FILE ISRAND ISAM IO RANDOM SEARCHKEY
           DISK 3340 STATUS RANDFLAG
           FB 40 1600 KEYLEN 5 KEYLOC 2
           CYLOFL 3 EXTENTS 2 (optional parameters)
           RANDREC 40 KILLIT 1 1

(z/OS) FILE MSTRFL ISAM FB 40 STATUS MASTFLAG
       FB 40 KEYLEN 5
       MASTREC 40 MASTEMPL 5 2

       FILE ISRAND ISAM IO RANDOM SEARCHKEY
       FB 40 KEYLEN 5 STATUS RANDFLAG
       RANDREC 40 KILLIT 1 1
```

Figure 534 ISAM Sequential Update

The first file definition describes the [ISAM](#) file as a sequential input file. The second file definition describes the same file as a random input/output file. Two different file names must be used, pointing to the same [ISAM](#) file, and JCL must be supplied for each file name.

The following keywords are required:

IO	Allows the file to be updated in place. The record format and record size must be specified on the FILE statement in conjunction with this keyword.								
RANDOM dataname	Any name not defined elsewhere in your program. Its length is equal to the key length. It holds the key of the record to be retrieved. A CH (character) data type is assumed. If your key is signed numeric, packed decimal, or binary, code NU, PD, or BI respectively, after dataname: <pre>RANDOM dataname PD</pre> Also, if the number of decimal places is other than zero, code it after the data type: <pre>RANDOM dataname PD 2</pre>								
KEYLEN nnn	Specifies the length of the record key. The allowable values for nnn are 1 to 255.								
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: <table><tr><td>blank</td><td>No record found.</td></tr><tr><td>Y</td><td>Record found.</td></tr><tr><td>E</td><td>End of file.</td></tr><tr><td>I</td><td>Duplicate or invalid key.</td></tr></table>	blank	No record found.	Y	Record found.	E	End of file.	I	Duplicate or invalid key.
blank	No record found.								
Y	Record found.								
E	End of file.								
I	Duplicate or invalid key.								

This technique requires that the [ISAM](#) file be read sequentially using the MSTRFL file. When a record is found that needs to be changed or added to the file, that record is read using the ISRAND file. The required change is made in the record area of the random file, and the record is then written (back) to the ISRAND file.

For example, you have an [ISAM](#) employee master file and you want to implement a flat 9% raise for all employees hired before Oct. 15, 1995. In addition, you want to delete the master record of any employee who was terminated before Apr. 15, 1995.

The following procedure logic can be coded:

```
:  
:  
READMSTR:  
  READ MSTRFL  
  IF MASTFLAG EQ 'E' STOP ENDIF  
COMPARE:  
  IF (TERM_DATE NE 0 AND TERM_DATE LT 950415)  
    GOTO UPDATE ENDIF  
  IF (HIRE_DATE LT 951015 AND TERM_DATE EQ 0)  
    GOTO UPDATE ENDIF
```

Figure 535 Procedure Logic Code (Page 1 of 2)

```

      GOTO READMSTR
UPDATE:
      MOVE MASTEMPL TO SEARCHKEY
      READ ISRAND
      IF RANDFLAG NE 'Y' GOTO KEYERROR ENDIF
      IF TERM_DATE NE 0 MOVE X'FF' TO KILLIT
      GOTO REWRITE  ENDIF
      RATE = RATE * 1.09
REWRITE:
      WRITE ISRAND
      GOTO READMSTR
      .
      .

```

Figure 535 Procedure Logic Code (Page 2 of 2)

New records can be inserted at any time. Move the key of the new record to the RANDOM dataname (SEARCHKEY) and read the ISRAND file. Check to make sure that the record is NOT found (RANDFLAG NE Y). Then, build the new record in the random file record area (RANDREC) and issue a WRITE command to add the record.

When the MSTRFL file reaches the end, the program issues a STOP command. You must always specify when to end the run if a RANDOM update is used.

To update variable-length records, see [Considerations for Processing Variable-Length Records on page 556](#).

z/OS Considerations

If OPTCD=L was specified as part of the DCB= parameter in the JCL when the [ISAM](#) file was created, the system does not read any records with X'FF' in the first byte. Also, if the addition of a record to the file results in the system moving records from the prime data area into an overflow area, any records having X'FF' in the first byte are dropped.

VSE Considerations

DISK and KEYLOC are additional required parameters.

DISK nnnn Defines the type of disk that contains the file. The allowable values are 2311, 2314, 3330, 3340, and 3350.

KEYLOC nnnn Specifies the starting location of the key within the data portion of the record. The allowable values of nnnn are 1 to 4095.

The following additional keywords are optional for VSE:

CYLOFL nn Where nn is the number of tracks reserved on each cylinder for cylinder overflow. The default is 0. This parameter must be supplied if cylinder overflow is specified when the file is initially loaded.

MASTER This keyword specifies that a master index is allocated for this file when it is loaded.

EXTENTS nn Where nn is the total number of extents allocated to the file. The default is 3 (cylinder index, prime data, and independent overflow). The value for nn must be between 2 and 20.

Each extent is identified in the JCL by an extent type code:

master index 4

cylinder index 4

prime data 1

independent overflow 2

Each extent is further numbered from 1 up (unless there is a master index that has an extent sequence number of 0).

Typical JCL for reading an [ISAM](#) file:

```
// ASSGN SYS005,X'190'  
// ASSGN SYS006,X'191'  
// DLBL MSTFRL,'MY.FILE.IDENT',90/365,ISE  
// EXTENT SYS005,111111,4,1,... Cylinder index  
// EXTENT SYS005,111111,1,2,... Prime data  
// EXTENT SYS006,123456,2,3,... Independent overflow
```

Figure 536 Typical JCL for Reading an ISAM File

The ISE subparameter on the DLBL statement must be specified when reading or updating an [ISAM](#) file.

VSE [ISAM](#) files having one record to a block (unblocked) require special consideration. If you need to update this type of file, see [Unblocked ISAM Files in VSE Considerations on page 558](#).

Random Update

This section describes the random insertion, change, and deletion of records on an [ISAM](#) file. The file is updated in place, without copying the old file to a new file.

Note: This type of run is not restartable. Once one record has been added, changed, or deleted, the file is not the same, regardless of why the job ended. Before running a program that uses this technique, design a method of recovering the original file.

For [ISAM](#) files, you can add new records, change existing records, and flag records for deletion. Generally speaking, deleted records remain physically in the file until removed by a backup and restore (re-org) run.

To update a file, describe the file to the operating system using JCL.

```
(VSE)      // ASSGN SYSnnn,X'cuu'
           // DLBL MSTRFL,'MY.FILE.IDENT',    ...
           // EXTENT SYSnnn,    ...

(z/OS) //MSTRFL DD DSN=MY.FILE.IDENT,DISP=OLD
```

Figure 537 Random Update

Note the disposition of OLD. This choice (also available under VSE, Levels 2 and up) prevents other users from accessing the file while it is being updated by your program.

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

ISAM Random Update

Use VISION:Results statements to define the input file:

```
(VSE)      FILE MSTRFL ISAM IO RANDOM SEARCHKEY
           DISK 3340 FB 40 1600 STATUS STATFLG
           KEYLEN 5 KEYLOC 2
           CYLOFL 3 EXTENTS 2 (optional parameters)
           MASTREC 40 KILLIT 1 1

(z/OS) FILE MSTRFL ISAM IO RANDOM SEARCHKEY
       FB 40 KEYLEN 5 STATUS STATFLG
       MASTREC 40 KILLIT 1 1
```

Figure 538 ISAM Random Update

The following keywords are required:

IO	Allows the file to be updated in place. The record format and record size must be specified on the FILE statement in conjunction with this keyword.
RANDOM dataname	Any name not defined elsewhere in your program. Its length is equal to the key length. It holds the key of the record to be retrieved. A CH (character) data type is assumed. If your key is signed numeric, packed decimal, or binary, code NU, PD, or BI, respectively, after dataname: RANDOM dataname PD Also, if the number of decimal places is other than zero, code it after the data type: RANDOM dataname PD 2

STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: blank No record found. Y Record found. E End of file. I Duplicate or invalid key.
KEYLEN nnn	Specifies the length of the record key. The allowable values for nnn are 1 to 255.

For example, you have an [ISAM](#) file that uses an employee number as its key. You also have a transaction file with records in no particular sequence that have the same format as records on the [ISAM](#) file. If the transaction employee number (EMPLOYEE) is not on the [ISAM](#) file, the employee is to be added. If the employee transaction matches a record on the [ISAM](#) file, you want to replace the [ISAM](#) record or flag it for deletion, if the employee is terminated.

The following procedure logic can be coded:

```
:
:
READTRAN:
  READ TRANS
  IF TRANSEOF EQ 'E' STOP ENDIF
  MOVE EMPLOYEE TO SEARCHKEY
  READ MSTRFL
  IF STATFLG NE 'Y' GOTO ADDRTN ENDIF
  IF TERMINATE EQ 'Y' MOVE X'FF' TO KILLIT
  GOTO REWRITE ENDIF
  MOVE TRANREC TO MASTREC
REWRITE:
  WRITE MSTRFL
  GOTO READTRAN
ADDRTN:
  MOVE TRANREC TO MASTREC
  WRITE MSTRFL
  GOTO READTRAN
```

Figure 539 Procedure Logic Code

When the TRANS file reaches the end, the program issues a STOP command. You must always specify when to end the run if an IO RANDOM update is used.

Before a record can be replaced, it must be read and found. Before a record can be added, it must be read and not found, even if you know it is not there.

To update variable-length records, see [Considerations for Processing Variable-Length Records on page 556](#).

The MSTRFL filename must be the same as the file name or ddname specified in the previous JCL.

VSE Considerations

DISK and KEYLOC are additional required parameters.

DISK nnnn	Defines the type of disk that contains the file. The allowable values are 2311, 2314, 3330, 3340, and 3350.
KEYLOC nnnn	Specifies the starting location of the key within the data portion of the record. The allowable values of nnnn are 1 to 4095.

The following additional keywords are optional for VSE:

CYLOFL nn	Where nn is the number of tracks reserved on each cylinder for cylinder overflow. The default is 0. This parameter must be supplied if cylinder overflow is specified when the file is initially loaded.								
MASTER	This keyword specifies that a master index is allocated for this file when it is loaded.								
EXTENTS nn	Where nn is the total number of extents allocated to the file. The default is 3 (cylinder index, prime data, and independent overflow). The value for nn must be between 2 and 20. Each extent is identified in the JCL by an extent type code: <table> <tr> <td>master index</td><td>4</td></tr> <tr> <td>cylinder index</td><td>4</td></tr> <tr> <td>prime data</td><td>1</td></tr> <tr> <td>independent overflow</td><td>2</td></tr> </table> Each extent is further numbered from 1 up (unless there is a master index that has an extent sequence number of 0).	master index	4	cylinder index	4	prime data	1	independent overflow	2
master index	4								
cylinder index	4								
prime data	1								
independent overflow	2								

Typical JCL for updating an [ISAM](#) file:

```
// ASSGN SYS005,X'190'
// ASSGN SYS006,X'191'
// DLBL MSTRFL,'MY.FILE.IDENT',90/365,ISE
// EXTENT SYS005,111111,4,1,... Cylinder index
// EXTENT SYS005,111111,1,2,... Prime data
// EXTENT SYS006,123456,2,3,... Independent overflow
```

Figure 540 Typical JCL for Updating an ISAM File

The ISE subparameter on the DLBL statement must be specified when updating an [ISAM](#) file.

VSE [ISAM](#) files having one record to a block (unblocked) require special consideration. If you need to update this type of file, see [Unblocked ISAM Files in VSE Considerations on page 558](#).

Sequential Insert or Extend

This section describes the writing of additional records to the end of any [ISAM](#) file.

To extend a file, describe the file to the operating system using JCL.

```
(VSE)      // ASSGN SYSnnn,X'cuu'
           // DLBL MYFILE,'MY.FILE.IDENT',    ...
           // EXTENT SYSnnn,    ...

(z/OS) //MYFILE DD DSN=MY.FILE.IDENT,DISP=    ...
```

Figure 541 Sequential Extend

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

ISAM Sequential Extend

This method of adding records to an [ISAM](#) file should not be used if you need to extend the file by more than a few records. The IBM [ISAM](#) I/O module must read through the indexes for the entire file each time it adds a record whose key is higher than any key in the file. The process takes longer and longer as each new record is added to the end.

You also can sort the input file in descending order by key and add the records to the file using the [Random Update](#) feature; or you can copy the [ISAM](#) file to a sequential file, add the new records to the end of that file, and recreate the [ISAM](#) file from the composite file (see [Creating a File on page 536](#)).

Use VISION:Results statements to define the input file and the file to be extended (output):

```
(VSE)      FILE TAPEIN FB 40 4800
           FILE MYFILE ISAM OUTPUT FROM TAPEIN DISK 3340
           FB 40 1600 KEYLEN 5 KEYLOC 2
           CYLOFL 3 EXTENTS 2 (optional parameters)

(z/OS) FILE TAPEIN FB 40 4800
       FILE MYFILE ISAM OUTPUT FROM TAPEIN
       FB 40 1600 KEYLEN 5 KEYLOC 2
```

Figure 542 ISAM Sequential Extend

KEYLEN nnn	Specifies the length of the record key. The allowable values of nnn are 1 to 255. This is a required parameter for VSE systems.
KEYLOC nnnn	Specifies the starting location of the key within the data portion of the record. The allowable values of nnnn are 1 to 4095. This is a required parameter for VSE systems.

To handle variable-length records, see [Considerations for Processing Variable-Length Records on page 556](#).

The FILE statements above read each record on the input file and add it to the end of the output file, unchanged. For information about defining fields (data items), see *Advantage VISION:Results for z/OS Getting Started*. For information about record selection and data manipulation, see [Chapter 7: Using the IF Command](#) and [Chapter 10: Using the MOVE Command](#).

The records on the input file must be in sequence by key for this run. In other words, each record written to the output file must have a key higher than the key in the previous record. Duplicates are not allowed. All records on the input file must have a key higher than the highest key on the [ISAM](#) file.

The MYFILE filename must be the same as the file name or ddname specified in the previous JCL.

z/OS Considerations

Specify DISP=MOD on the DD statement that describes the [ISAM](#) file in the JCL.

VSE Considerations

The DISK parameter is required and is coded as follows:

DISK nnnn Defines the type of disk that contains the file. The allowable values are 2311, 2314, 3330, 3340, and 3350.

The following additional keywords are optional for VSE:

CYLOFL nn Where nn is the number of tracks reserved on each cylinder for cylinder overflow. The default is 0. An independent overflow area, also used to hold inserted records, can be used in addition to or instead of cylinder overflow.

MASTER This keyword specifies that a master index was allocated for this file.

EXTENTS nn Where nn is the total number of extents allocated to the file. The default is 3 (cylinder index, prime data, and independent overflow). The value of nn must be between 2 and 20.

Each extent is identified in the JCL by an extent type code:

master index	4
cylinder index	4
prime data	1
independent overflow	2

Each extent is further numbered from 1 up (unless there is a master index that has an extent sequence number of 0).

Typical JCL for an [ISAM](#) file extend run is:

```
// ASSGN SYS005,X'190'  
// ASSGN SYS006,X'191'  
// DLBL MYFILE,'MY.FILE.IDENT',90/365,ISE  
// EXTENT SYS005,111111,4,1,... Cylinder index  
// EXTENT SYS005,111111,1,2,... Prime data  
// EXTENT SYS006,123456,2,3,... Independent overflow
```

Figure 543 Typical JCL for an ISAM File Extend Run

The ISE subparameter on the previous DLBL statement must be specified when extending an [ISAM](#) file.

VSE [ISAM](#) files having one record to a block (unblocked) require special consideration. If you need to extend this type of file, see [Unblocked ISAM Files in VSE Considerations on page 558](#).

Considerations for Processing Variable-Length Records

VISION:Results simplifies the processing of files with variable-length records and spanned variable-length records.

Record Size

When specifying the logical record length for a variable file, you must specify the maximum data length. VISION:Results adjusts this length to allow for the record descriptor word (RDW).

Block Size

When specifying the maximum block size for [ISAM](#) files, specify the actual physical size of the block. In calculating this, add 4 to the specified record length to allow for the RDW, multiply by the blocking factor, and then add 4 to allow for the block descriptor word (BDW).

For example, a variable file, containing records with a maximum length of 200, that is to be blocked 10 has a maximum block size of 2044 $((200+4)*10+4)$. The maximum block size specified for a variable file does not have to allow for a fixed number of the maximum length records. The specific application determines the optimum block size. However, VISION:Results requires that the specified block size be at least 8 greater than the record size given.

Location of Data Within Record

VISION:Results assumes that position 1 of every variable record is the first byte of data. It automatically takes care of the 4-byte RDW for non-VSAM variable files. Users setting up data definitions for variable-length records should start defining fields from the first byte of data.

Length of Current Record

On input, VISION:Results keeps track of the length of the data in the record just read. You can access this information by specifying the keyword `LENGTH`. A data name specified this way is a 2-byte binary field and contains the length of the data in the record just read.

For output files, you must specify how long the data portion of the current record is. This length is used to build an RDW before writing the record.

Output from Itself

If a file is defined to be output from itself, it has its own I/O area assigned. It also has a 2-byte binary length field assigned to the file. Reference this length field by specifying the keyword `LENGTH` followed by a data name. It is your responsibility to move data to the area and place the length of the data in the `LENGTH` dataname before the record is to be written, either using the automatic cycle or a `WRITE` command. Failure to move a valid length to the `LENGTH` dataname may result in wrong length records being written.

Output from an Input file

When a file is written from an input file, VISION:Results does not assign an I/O area for the file or a length field. VISION:Results references the I/O area and length field for the input file when the output file is written. VISION:Results places restrictions on the type of file from which a variable output file can be written. A variable [ISAM](#) file can be written only from another variable non-VSAM file. If other combinations are required, you must specify `OUTPUT` from itself for the file, move the data, and set up the data length.

When a record is being written from an input area, the length is automatically set up by VISION:Results when the record is read. `LENGTH` cannot be specified for the output file because the length field does not exist for this file. You can specify `LENGTH` dataname for the input file if you want to adjust the length of the record. You can shorten a record to be written by changing the contents of the `LENGTH` dataname. Do not attempt to write a longer record. If it is necessary to increase the length of a record, specify `OUTPUT` from itself on the `FILE` statement, move the data, and set up the length. The `LENGTH` dataname always contains the length of the data only.

File Printing of Variable Files

When VISION:Results performs a file print of the record, either using the OPTION statement or one of the file print commands (HEXPRINT, PRINT, and so on), the RDW is printed if it is present. This helps programmers in debugging. The data that is defined at byte 1 in your data definition is at byte 5 in the file print of your record if there is an RDW. This is also true for the dump of the record in a PRINTERERROR error analysis dump.

Sorting a Variable File

To eliminate a possible error during SORT, variable-length records are moved from the I/O area to a fixed-length area for processing. If an error occurs in the execution, the record shown in the I/O area in the PRINTERERROR dump is not necessarily the record you are processing. The variable record has been moved to a work area. Check your record there.

z/OS users must specify variable record format in their FILE statement for an [ISAM](#) file to have VISION:Results handle the RDW automatically. If the record format is not specified at compile time, VISION:Results passes you the ENTIRE record (including the RDW) in the record area.

Unblocked ISAM Files in VSE Considerations

Blocked [ISAM](#) files and unblocked [ISAM](#) files are handled differently by the IBM VSE I/O routines. The following sections outline the differences and provide guidelines for processing unblocked [ISAM](#) files.

Sequential or Skip Sequential Read

When a blocked [ISAM](#) file is read, the system brings in only the data record (not the key prefix). When an unblocked [ISAM](#) file is read sequentially, the key precedes the data in the record area even though the key can also be embedded in the record.

Example

Read an unblocked [ISAM](#) file whose record length is 35 and key length is 5.

```
FILE ISAMIN F 35 ISAM DISK 3340 KEYLEN 5 KEYLOC 1
FIRSTBYTE 1 6
NEXTFLD 10 7
```

Figure 544 Sequential or Skip Sequential Read

The first byte of data is located in byte 6 of the record area. Also, note that the last 5 bytes of data are beyond the end of the record area. If you need to access the last n bytes of the record, where n is the key length, set up a work area and a variable length to move it and to obtain the entire record:

```
MOVE FIRSTBYTE LENGTH 35 TO HOLDREC
```

Figure 545 Move First Byte of Data

Sequential Load or Extend

Unblocked [ISAM](#) sequential output requires that the record be written from an area containing the key followed by the data.

To create an unblocked [ISAM](#) file with a record length of 35 and a key length of 5:

```
FILE ISAMOUT F 35 ISAM DISK 3340  
  KEYLEN 5 KEYLOC 1 OUTPUT FROM WORKFL  
FILE WORKFL F 40 TAPE NL SYS029  
  WORKKEY 5  
  WORKREC 35
```

Figure 546 Sequential Load or Extend

The file WORKFL is not a real file, just an area from which to write. In this example, SYS029 must be assigned to IGN (ignore) in the JCL. The key of the output record is moved to WORKKEY and the data to WORKREC.

Random Retrieval

Random retrieval of unblocked [ISAM](#) files works the same way as blocked [ISAM](#). Move the key you are searching for to your RANDOM dataname and issue a read. The data is returned in your record area without the key prefix. Byte 1 of the data is in byte 1 of the record area.

Random Update

In this type of access, the record must be randomly retrieved first (see [Unblocked ISAM Files in VSE Considerations on page 558](#)). If the record is found (STATUS dataname = Y), the replacement record only is moved to the output area. If the record is not found, the key and data are moved to the output area. In both cases, a WRITE command is issued to replace or add the record as necessary.

```
FILE ISAMUPD F 35 ISAM DISK 3340 RANDOM IKEY
  KEYLEN 5 KEYLOC 1 IO FROM WORKFL
RECIN 35
FILE WORKFL F 40 TAPE NL SYS029
  OLDREC 35 1
  NEWKEY 5 1
  NEWREC 35 6
```

Figure 547 Random Update

Remember to assign SYS029 to IGN, as before.

Numeric and Packed Key Considerations

When processing [ISAM](#) files where you are building records or retrieving records in random or skip mode, consider the sign of the key if the key has a data type of numeric (zoned decimal) or packed. A positive sign for a numeric or packed field can be C or F. This can be a problem if you are searching for a particular record on an [ISAM](#) file and the key you are using to search has a different sign than the key in the record. For example, a 2-byte packed field with the value 123C is not considered equal to a 2-byte packed field with the value 123F. A NO RECORD FOUND status is set.

You can run into an unlike sign problem under the following circumstance: a packed key in the [ISAM](#) key area has developed a C sign (perhaps the key had been converted from numeric to packed and a C sign resulted from this process). If the key in the record contains an F sign, a status code of blank (no record found) is returned to the program after a random retrieval read. The same situation can occur with numeric fields.

To prevent or correct for unlike signs

1. Whenever possible, treat the keys as character data:

```
FILE ISAMIN F 120 RANDOM SEARCHKEY
  STATUS STATUSFLAG KEYLEN 3
FILE TRANIN STATUS TRNEOF
  KEYIN 3 CH ; (key is actually packed decimal)
PROCEDURE:
  IF TRNEOF EQ 'E' STOP ENDIF
  MOVE KEYIN TO SEARCHKEY ; (sign remains unchanged)
  READ ISAMIN
  IF STATUSFLAG NE 'Y' GOTO NOTFOUND ENDIF
  .
  .
```


The keys actually have a data type of packed, but are defined as character, so no sign change can occur. If the key has to be treated as packed for other purposes, you can redefine it.

To convert a packed key's sign from C to F, you can use OR on the field. In [Figure 548](#), a 2-byte packed key is having its sign set to F.

```
COMBINE BITS X'000F' OR SRCHKEY
```

Figure 548 Convert a Packed Key's Sign From C to F

To convert a packed key's sign from F to C, you can use AND on the field. In [Figure 549](#), a 2-byte packed key with an F sign is having its sign set to C.

```
COMBINE BITS X'FFFC' AND SRCHKEY
```

Figure 549 Convert a Packed Key's Sign From F to C

ISAM Examples

Example 1 Fixed-Length ISAM In, ISAM Out

The entire [ISAM](#) file is copied to a new [ISAM](#) file.

```
FILE ISAMIN ISAM FB 140 1400 DISK 3330
  KEYLEN 7 KEYLOC 3 CYLOFL 2 EXTENTS 2
FILE ISAMOUT ISAM OUTPUT FROM ISAMIN FB 140 1400
  DISK 3330 KEYLEN 7 KEYLOC 3 CYLOFL 4 EXTENTS 3
```

Figure 550 [Example 1](#) Fixed-Length ISAM In, ISAM Out

Example 2 Variable-Length ISAM In, Variable-Length File Out (z/OS only)

The entire [ISAM](#) file is unloaded to tape.

Record and block sizes can be specified in the JCL.

```
FILE ISAMIN ISAM VB 120
FILE TAPEOUT OUTPUT FROM ISAMIN VB 120 2484
```

Figure 551 [Example 2](#) Variable-Length ISAM In, Variable-Length File Out (z/OS only)

Example 3 Variable-Length File In, ISAM Out (z/OS only)

The entire tape file is copied to create a new [ISAM](#) file.

The JCL for the [ISAM](#) file should indicate DISP=NEW.

Record and block size for the tape file is obtained from the tape label.

All characteristics of the [ISAM](#) file (except RECFM) are defined in the JCL.

```
FILE TAPEIN VB
FILE ISAMOUT ISAM VB OUTPUT FROM TAPEIN
```

Figure 552 [Example 3](#) Variable-Length File In, ISAM Out (z/OS only)

Example 4 Fixed-Length File In, ISAM Out, with SORT

The entire input file is sorted by key. The sorted records are then used to load a new [ISAM](#) file.

For z/OS, the JCL for the [ISAM](#) file must specify DISP=(NEW, and so on). For VSE, the DLBL must indicate a file type of ISC.

No JCL is supplied for SORTIN or SORTOUT; however, all other JCL normally required for a sort is needed.

```
FILE TAPEIN SYS041 FB 95 2850 TAPE
  CUSTNUM 9 2
FILE ISAMOUT ISAM OUTPUT FROM TAPEIN FB 95 950
  DISK 3340 KEYLEN 9 KEYLOC 2 EXTENTS 2 CYLOFL 1

SORT TAPEIN USING CUSTNUM WORK 3
```

Figure 553 [Example 4](#) Fixed-Length File In, ISAM Out, with SORT

Example 5 Skip Sequential Read Fixed-Length ISAM

A generic search by class code is performed.

The 8-byte key (in the WORKAREA) is padded with binary 0s to guarantee that processing begins at the first record for any given class.

All records found for each class on the FINDERS file are written to the report. Each class commences on a new page.

```
FILE INVMAS ISAM SKIP PRODCODE STATUS SRCHFLAG
  DISK 3330 FB 75 1500 KEYLEN 8 KEYLOC 3
MSTCLASS 2 3 (CLASS)
MSTPRODUCT 6 (PRODUCT'CODE)
QTYONHAND 5 29 PD Z (ON HAND)
QTYORDRD 5 42 PD Z (QUANTITY'ORDERED)
ORDRDATE 6 47 NU D (DATE'ORDERED)

FILE FINDERS SYS007 FB 80 1600 STATUS ENDFLAG
  CLASSCODE 2

WORKAREA
  SRCHCLASS 2
  FILLER 6 VALUE X'00000000000000'

IF ENDFLAG EQ 'E' STOP ENDIF
MOVE CLASSCODE TO SRCHCLASS
MOVE SRCHCLASS TO PRODCODE LENGTH 8
READINV:
  READ INVMAS
```

Figure 554 [Example 5](#) Skip Sequential Read Fixed-Length ISAM (Page 1 of 2)

```

IF SRCHFLAG EQ 'E' REJECT ENDIF
IF MSTCLASS GT CLASSCODE MOVE 66 TO DYLLINE
REJECT ENDIF
LIST SUPPRESS MSTCLASS MSTPRODUCT ORDRDATE
QTYORDRD QTYONHAND
GOTO READINV

T1 'INVENTORY REPORT FOR SELECTED PRODUCT CLASSES' WITH 2 AFTER
T1+120 DYLDATE

```

Figure 554 [Example 5](#) Skip Sequential Read Fixed-Length ISAM (Page 2 of 2)

Example 6 Random Read Fixed-Length ISAM

A random search of a master file (CUSMSTR) is performed to obtain the customer's name for a sales report line.

```

FILE SALES TAPE SYS035 FB 128 1280 STATUS ALLDONE
SALESMAN 3 3
CUSTNUM 6 26
PRODUCT 7 32
QUANTITY 3 54 PD Z
UNITPRICE 4 81 PD 2 E

FILE CUSMSTR ISAM RANDOM FINDCUST STATUS MSTRSTAT
DISK 3340 FB 225 900 KEYLEN 6 KEYLOC 2 CYLOFL 2 EXTENTS 2
CUSTOMER 25 8

WORKAREA
CNAME 25 (CUSTOMER NAME)
TOT 7 PD 2 E
TALLY E 3

IF ALLDONE EQ 'E' STOP ENDIF
MOVE CUSTNUM TO FINDCUST
READ CUSMSTR
IF MSTRSTAT EQ 'Y' MOVE CUSTOMER TO CNAME
ELSE MOVE '*** NOT ON FILE ***' TO CNAME ENDIF
TOT = QUANTITY * UNITPRICE
LIST SUPPRESS SALESMAN CNAME PRODUCT UNITPRICE QUANTITY TOT
CONTROL SALESMAN
ON CHANGE IN SALESMAN
LIST 'TOTAL SALESMAN' AT CNAME TALLY SUM TOT WITH 2 BEFORE
AND EJECT AFTER
ON FINAL LIST '*** FINAL TOTAL ***' AT PRODUCT SUM TOT
T1 'S A L E S B Y S A L E S M A N' WITH 2 AFTER
T1+100 DYLDATEPAG

```

Figure 555 Example 6 Random Read Fixed-Length ISAM

Example 7 Random Update Fixed-Length ISAM

An [ISAM](#) file is updated randomly by adding new records, replacing existing records, and flagging others for deletion.

For z/OS, if OPTCD=L was specified as a DCB subparameter when the [ISAM](#) file was created, the X'FF' placed in the first byte of a deleted record causes the access method to ignore that record when reading the file and drop the record while moving data to an overflow area.

For VSE, subsequent programs must check the MASTDEL byte themselves, after reading a record, to find out if the record has been deleted.

```
FILE ACCTMST ISAM IO RANDOM ACCTKEY
  DISK 3330 FB 80 800 STATUS MSTFLAG
  KEYLEN 6 KEYLOC 5 CYLOFL 2 EXTENTS 4
MASTREC 80
MASTDEL 1 1

FILE TRANS DISK 2311 FB 80 3600 STATUS TRANSEOF
  TRANREC 80
  TRANCODE 1 1
  TRANACCT 6 5

READTRAN:
  READ TRANS
  IF TRANSEOF EQ 'E' STOP ENDIF
  MOVE TRANACCT TO ACCTKEY
  READ ACCTMST
  IF TRANCODE EQ 'A' GOTO ADDRTRN ENDIF
  IF TRANCODE EQ 'D' GOTO DELRTN ENDIF
CHGRTRN:
  IF MSTFLAG NE 'Y' MOVE 'BAD CHANGE' TO DYLPRTCOMM
  PRINT TRANREC GOTO READTRAN ENDIF
  MOVE TRANREC TO MASTREC
  GOTO WRITEMST
ADDRTRN:
  IF MSTFLAG EQ 'Y' MOVE 'BAD ADDN' TO DYLPRTCOMM
  PRINT TRANREC PRINT MASTREC
  GOTO READTRAN ENDIF
  MOVE TRANREC TO MASTREC
  GOTO WRITEMST
DELRTN:
  IF MSTFLAG NE 'Y' MOVE 'BAD DELETE' TO DYLPRTCOMM
  PRINT TRANREC GOTO READTRAN ENDIF
  MOVE X'FF' TO MASTDEL
WRITEMST:
  WRITE ACCTMST
  GOTO READTRAN
```

Figure 556 [Example 7](#) Random Update Fixed-Length ISAM

Chapter 22: BDAM Processing

VISION:Results supports Basic Direct Access Method (BDAM) files with fixed, variable, or undefined record types. The records can be processed either sequentially or randomly. The use of relative track and actual key, relative track and actual record number, or relative block (z/OS only) addressing schemes is also supported with either sequential or random access methods.

Direct access files are processed using BDAM in z/OS and DAM in VSE. In this chapter, whenever the term BDAM is used, it also refers to DAM, unless indicated otherwise.

Syntax Format

The keywords and operands pertaining only to BDAM files are described in this chapter. For details about operands such as filename, F, V, U, recordsize, and so on, see [Chapter 6: Using the FILE Command](#).

```
FILE filename
BDAM {RELTRK dataname2 | RELBLK dataname2}
STATUS dataname2
{F | V | U} recordsize
{INPUT | OUTPUT FROM filename | IO}
[RANDOM dataname2 {CH | NU | BI | PD} [decimals] |
  SEQUENTIAL [dataname2 {CH | NU | BI | PD} [decimals]]}
[KEYLEN nnn] [COUNT dataname2] [LENGTH dataname2]
[DISK {2311 | 2314 | 3330 | 3340 | 3350}]
[SYSnnn] [EXTENTS nn]
```

Figure 557 BDAM Syntax Format

Record Addressing Schemes

The three types of addressing schemes are:

- Relative track and actual key.
- Relative track and actual record number.
- Relative block number (z/OS only).

Relative Track and Actual Key

Fixed, variable, and undefined length records can be used. Specify the record address as a 2-byte binary relative track number (the first track number is 0) and a 1- to 255-byte key. The following FILE statement keywords indicate this addressing scheme:

BDAM	Indicates BDAM file processing.
SEQUENTIAL dataname or RANDOM dataname	Any name not defined elsewhere in your program. It holds the record key. Its length is equal to the key length. A CH (character) data type is assumed. If your key is signed numeric, packed decimal, or binary, you must code NU, PD, or BI, respectively, after dataname: SEQUENTIAL dataname PD Also, if the number of decimal places is other than zero, code it after the data type: RANDOM dataname PD 2
KEYLEN n	Indicates the length of the key. Acceptable values for n are 1 to 255.
RELTRK dataname	Any name not defined elsewhere in your program. It holds the relative track number and is a 2-byte binary field.

Relative Track and Actual Record Number

Fixed, variable, and undefined length records can be used; however, variable and undefined length records cannot be added to a file during random creation or random update. Specify the record address as a 3-byte field containing a 2-byte relative track number (the first track number is 0), followed by a 1-byte binary record number (the first record number is 1). The following FILE statement keyword indicates this addressing scheme:

BDAM	Indicates BDAM file processing.
RELTRK dataname	Any name not defined elsewhere in your program. It holds the relative track number and actual record number. It is a 3-byte binary field.

Relative Block

The relative block addressing scheme is available in z/OS only. Fixed-length records must be used—keys are not allowed. Specify the relative block number as an ordinal number (the first block number is 0). The following FILE statement keyword indicates this addressing scheme:

BDAM	Indicates BDAM file processing.
RELBLK dataname	Any name not defined elsewhere in your program. It holds the relative block number. It is a 3-byte binary field.

Access Methods

The two types of access methods are:

- Sequential
- Random

Sequential Access

This access method allows you to create and retrieve a direct access file using fixed, variable, and undefined length records both with and without keys. The following FILE statement keyword indicates this access method:

SEQUENTIAL dataname	The dataname is required only if the key is used (see Record Addressing Schemes on page 566).

Random Access

This method allows you to create, retrieve, update, and add fixed, variable, and undefined length records with a key, and fixed-length records without a key. Variable and undefined length records without a key can be retrieved and replaced only. The following FILE statement keyword indicates this access method:

RANDOM dataname	The dataname is required only if a key is used (see Record Addressing Schemes on page 566).

I/O Statistics

VISION:Results maintains a count of records read or written. To use this count, the following FILE statement keyword must be used:

COUNT dataname	Any name not defined elsewhere in your program. It is a 4-byte binary field.

For output files, this count is the number of records written. For input files, it is the number of records read. For input/output files, it is the number of records added or updated. Dummy records are not counted.

Random Create

This section describes the random creation of a new BDAM file or the replacement of an old one. If you intend to read or update an existing file only, you do not need to be concerned with this procedure.

To create a file, describe it to the operating system using JCL.

```
(VSE)    // ASSGN  SYSnnn,X'cuu'  
         // DLBL   filename,'DIRECT FILE',99365,DA  
         // EXTENT SYSnnn,...extent information  
  
(z/OS) //BDAMFILE DD DSN=BDAM.FILE.IDENT,DISP=...
```

Figure 558 Create a New BDAM File Using JCL

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

The file name in your VISION:Results FILE statement must be the same as the file name or ddname specified in the previous JCL.

When a BDAM file is being created randomly, all the tracks of the file are initialized at open time with dummy records or capacity records. The BDAM file should contain enough space to hold the maximum number of records expected in the file. The total size of the data set is determined by the initial space allocation, and that size cannot be changed unless the file is recreated.

For the statements used to create a BDAM file using the relative track number and actual key, relative track number and actual record number, or relative block number addressing scheme, see [Relative Track Number and Actual Key on page 568](#), [Relative Track Number and Actual Record Number on page 570](#), and [Relative Block Number on page 572](#).

Relative Track Number and Actual Key

Use VISION:Results statements to define the file to be created (output):

```
(VSE)    FILE DAFILE BDAM OUTPUT FROM DAFILE F 35  
         RANDOM KEYAREA RELTRK TRKNUMB KEYLEN 5  
         LENGTH BLENGTH STATUS BDAMSTAT DISK 3350  
         EXTENT 2 (optional)  
  
(z/OS) FILE DAFILE BDAM OUTPUT FROM DAFILE F 35  
         RANDOM KEYAREA RELTRK TRKNUMB KEYLEN 5  
         LENGTH BLENGTH STATUS BDAMSTAT
```

Figure 559 Random Create—Relative Track Number and Actual Key

Specify the record format (F, V, or U) and the record size.

The following keywords are required:

BDAM	Indicates BDAM file processing.
OUTPUT FROM filename	BDAM files can be output only from themselves.
RANDOM dataname	Any name not defined elsewhere in your program. Its length is equal to the key length. It holds the key of the record to be written. A character (CH) data type is assumed. If your key is signed numeric, packed decimal, or binary, code NU, PD, or BI, respectively, after dataname:

`RANDOM dataname PD`

Also, if the number of decimal places is other than zero, code it after the data type:

`RANDOM dataname PD 2`

RELTRK dataname	Any name not defined elsewhere in your program. It holds the relative track number. It is a 2-byte binary field.
KEYLEN n	Specifies the length of the record key. Allowable values for n are 1 to 255.
LENGTH dataname	Any name not defined elsewhere in your program. It holds the length of the record to be written. Required only for undefined and variable-length records.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a WRITE has been issued for this file: Y Record was written correctly. N Space was not found on the track. L Length is incorrect. I Invalid request.

DISK type	VSE only. Defines the type of disk that contains the file. Valid values are 2311, 2314, 3330, 3340, and 3350.

EXTENTS n	VSE only. Number of EXTENT statements in the JCL for this file. Default is 1.

Undefined length records must be greater than or equal to 8 bytes; otherwise, an L is returned in the STATUS dataname.

If you attempt to write a record whose key is already on the specified track or if you try to write a record outside of the file, an I is returned in the STATUS dataname.

The first byte of your key must not be high-values (X'FF'); otherwise, an I is returned in the STATUS dataname.

If you try to write a record on a track that is already full, an N is returned in the STATUS dataname.

To create variable-length records, see [Other Considerations on page 597](#).

The following procedure logic can be coded:

```
.  
.  
READLOOP:  
  READ TAPEIN  
  IF TAPESTAT EQ 'E' STOP ENDIF  
  MOVE EMPLNUMB TO KEYAREA  
  MOVE EMPLNUMB TO RAND WORK  
  PERFORM RANDOMIZE TO RAND_X  
  MOVE RAND WORK TO TRKNUMB  
  MOVE EMPLDATA TO BDAMREC  
  WRITE DAFILE  
  IF BDAMSTAT NE 'Y' GOTO ERRORTN ENDIF  
  GOTO READLOOP  
.  
.
```

Figure 560 Coding the Procedure Logic

in [Figure 560](#), each record in a file TAPEIN contains an employee number (EMPLNUMB) and employee data (EMPLDATA). The program:

- Moves the number to the RANDOM dataname as a key.
- Randomizes the number and moves the result to the RELTRK dataname as a relative track number.
- Moves the data to the BDAM file record area.
- Issues a WRITE for the BDAM file.

When the TAPEIN file reaches the end (STATUS dataname is equal to E), the program issues a STOP command. You must always specify when to end the run if a BDAM file is used.

Note: The first relative track number is 0.

Relative Track Number and Actual Record Number

This addressing scheme can be used for fixed-length records only, when randomly creating a file.

Use VISION:Results statements to define the file to be created (output):

```
(VSE) FILE DAFILE BDAM OUTPUT FROM DAFILE F 35  
      RANDOM RELTRK RELADDR  
      STATUS BDAMSTAT DISK 3350  
      EXTENTS 2 (optional)  
  
(z/OS) FILE DAFILE BDAM OUTPUT FROM DAFILE F 35  
        RANDOM RELTRK RELADDR STATUS BDAMSTAT
```

Specify the record format as F (fixed) and the record size.

The following keywords are required:

BDAM	Indicates BDAM file processing.
OUTPUT FROM filename	BDAM files can be output only from themselves.
RANDOM	No data name is required.
RELTRK dataname	Any name not defined elsewhere in your program. It holds the relative track number and the actual record number on that track. It is a 3-byte binary field.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a WRITE has been issued for this file: Y Record was written correctly. I Invalid request.
DISK type	VSE only. Defines the type of disk that contains the file. Valid values are 2311, 2314, 3330, 3340, and 3350.
EXTENTS n	VSE only. Number of EXTENT statements in the JCL for this file. Default is 1.

If an attempt is made to write to an existing record or to write a record beyond the end of a track or outside of the file, an I is returned in the STATUS dataname.

To create variable-length records, see [Other Considerations on page 597](#).

The following procedure logic can be coded:

```
.
.
WORKAREA
  ADDRESS  3  BI
    REDEF  AT  ADDRESS
  ADDR TT  2  BI
  ADDR_R   1  BI
.
.
READLOOP:
  READ TAPEIN
  IF TAPESTAT EQ 'E' STOP ENDIF
  MOVE EMPLNUMB TO RAND WORK1
  PERFORM RANDOMIZE1 TO RAND_X1
  MOVE RAND WORK1 TO ADDR TT
  MOVE EMPLNUMB TO RAND WORK2
  PERFORM RANDOMIZE2 TO RAND_X2
  MOVE RAND WORK2 TO ADDR_R
  MOVE ADDRESS TO RELADDR
  MOVE EMPLDATA TO BDAMREC
  WRITE DAFILE
  IF BDAMSTAT NE 'Y' GOTO ERRORITN ENDIF
  GOTO READLOOP
.
.
```

Figure 561 Coding the Procedure Logic

In [Figure 561](#), each record in a file TAPEIN contains an employee number (EMPLNUMB) and employee data (EMPLDATA). The program does the following:

- Randomizes the number and moves the result to the RELTRK dataname as a relative track number.
- Randomizes the number and moves the result to the RELTRK dataname as an actual record number on that track.
- Moves the data to the BDAM file record area.
- Issues a WRITE for the BDAM file.

When the TAPEIN file reaches the end (STATUS dataname is equal to E), the program issues a STOP command. You must always specify when to end the run if a BDAM file is used.

Note: The first relative track number is 0 and the first actual record number is 1.

Relative Block Number

This addressing scheme can be used for fixed-length records only. It is not available in VSE.

Use VISION:Results statements to define the file to be created (output):

```
(z/OS) FILE DAFILE BDAM OUTPUT FROM DAFILE F 35
      RANDOM RELBLK BLKNUMB STATUS BDAMSTAT
```

Figure 562 Random Create — Relative Block Number (z/OS only)

Note: This is available in the z/OS version of VISION:Results only.

Specify the record format as F (fixed) and the record size.

The following keywords are required:

BDAM

OUTPUT FROM filename BDAM files can be output only from themselves.

RANDOM No data name is required.

RELBLK dataname Any name not defined elsewhere in your program. It holds the relative block number. It is a 3-byte binary field.

STATUS dataname Any name not defined elsewhere in your program. It is a 1-byte field containing one of the following values after a WRITE has been issued for this file:

Y	Record was written correctly.
I	Invalid request.

If an attempt is made to write to an existing record or to write a record outside of the file, an I is returned in the STATUS dataname.

The following procedure logic can be coded:

```
.
.
READLOOP:
  READ TAPEIN
  IF TAPESTAT EQ 'E' STOP ENDIF
  MOVE EMPLNUMB TO RAND WORK
  PERFORM RANDOMIZE TO RAND X
  MOVE RAND WORK TO BLKNUMB
  MOVE EMPLDATA TO BDAMREC
  WRITE DAFILE
  IF BDAMSTAT NE 'Y' GOTO ERRORTN ENDIF
  GOTO READLOOP
.
.
```

Figure 563 Coding the Procedure Logic

In [Figure 563](#), each record in a file TAPEIN contains an employee number (EMPLNUMB) and employee data (EMPLDATA). The program does the following:

- Randomizes the number and moves the result to the RELBLK data name as a relative block number.
- Moves the data to the BDAM file record area.
- Issues a WRITE for the BDAM file.

When the TAPEIN file reaches the end (STATUS dataname is equal to E), the program issues a STOP command. You must always specify when to end the run if a BDAM file is used.

The first relative block number is 0.

Sequential Create

This section describes the sequential creation of a new BDAM file or the replacement of an old one. If you intend to read or update an existing file only, you do not need to be concerned with this procedure.

To create a file, describe it to the operating system using JCL.

```
(VSE)      // ASSGN  SYSnnn,X'cuu'
           // DLBL   filename,'DIRECT FILE',99365,DA
           // EXTENT SYSnnn,...extent information

(z/OS)     //BDAMFILE DD DSN=BDAM.FILE.IDENT,DISP=...
```

Figure 564 Sequential Create a File Using JCL

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

The file name in your VISION:Results FILE statement must be the same as the file name or ddname specified in the previous JCL.

When a file is being created, all records are written serially in the sequence in which they are moved into the output buffer. All records on the first track must be processed before records are written on the second or third track, and so on.

After the last record has been written, VISION:Results adds dummy records or writes a capacity record to complete the final track on which data was written. If z/OS, the rest of the tracks allocated to the file are no longer available to you. If VSE, dummy records or capacity records are written on the remainder of the tracks in the file.

For the statements used to create a BDAM file using the relative track number and actual key, relative track number and actual record number, or relative block number addressing scheme, see [Sequential Create on page 573](#).

Relative Track Number and Actual Key

Use VISION:Results statements to define the file to be created (output).

```
(VSE)  FILE DAFILE BDAM OUTPUT FROM DAFILE F 35
        SEQUENTIAL KEYAREA RELTRK TRKNUMB KEYLEN 5
        LENGTH BLENGTH STATUS BDAMSTAT DISK 3350
        EXTENT 2 (optional)

(z/OS) FILE DAFILE BDAM OUTPUT FROM DAFILE F 35
        SEQUENTIAL KEYAREA RELTRK TRKNUMB KEYLEN 5
        LENGTH BLENGTH STATUS BDAMSTAT
```

Figure 565 Sequential Create — Relative Track Number and Actual Key

Specify the record format (F, V, or U) and the record size.

The following keywords are required:

BDAM	Indicates BDAM file processing.
OUTPUT FROM filename	BDAM files can be output only from themselves.
SEQUENTIAL dataname	Any name not defined elsewhere in your program. Its length is equal to the key length. It holds the key of the record to be written. A character (CH) data type is assumed. If your key is signed numeric, packed decimal, or binary, code NU, PD, or BI, respectively, after dataname: <pre>SEQUENTIAL dataname PD</pre> Also, if the number of decimal places is other than zero, code it after the data type: <pre>SEQUENTIAL dataname PD 2</pre>
RELTRK dataname	Any name not defined elsewhere in your program. It holds the relative track number. It is a 2-byte binary field.
KEYLEN n	Specifies the length of the record key. Allowable values for n are 1 to 255.

LENGTH dataname	Required only for undefined and variable-length records. Any name not defined elsewhere in your program. It holds the length of the record to be written.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a WRITE has been issued for this file: Y Record was written correctly. N Space was not found in the file. L Length is incorrect. I Invalid request.
DISK type	VSE only. Defines the type of disk that contains the file. Valid values are 2311, 2314, 3330, 3340, and 3350.
EXTENTS n	VSE only. Number of EXTENT statements in the JCL for this file. Default is 1.

When records are written sequentially, you do not need to update the contents of the relative track address. It is updated as follows:

- Records are written on the first available track until space is no longer available. At such time, VISION:Results increments the relative track number by 1 and continues writing on the next track.
- The value of the relative track number used by the system is made available to you in the relative track address after the record is written.

To skip tracks, place the number of the track to advance to in the RELTRK dataname. For example, to write a record on track 4 before you have filled tracks 2 and 3, move 4 to the RELTRK dataname before issuing the WRITE.

Note: If you attempt to skip more tracks than are allocated, your program aborts.

Dummy records or capacity records are added to complete the intervening tracks.

If the initial relative track number value is not 0, VISION:Results completes the preceding tracks with dummy or capacity records.

The first byte of your key must not be high value (X'FF'); otherwise, an I is returned in the STATUS dataname.

Note: On a sequential create, VISION:Results does not prevent you from writing two or more records on the same track with the same key.

If you attempt to write a record on a previous track or outside of the file, an I is returned in the STATUS dataname.

If you attempt to write a record, when end of file is reached, an N is returned in the STATUS dataname.

Undefined length records must be greater than or equal to 8 bytes; otherwise, an L is returned in the STATUS dataname.

To create variable-length records, see [Other Considerations on page 597](#).

The following procedure logic can be coded:

```
.  
.   
READLOOP:  
  READ TAPEIN  
  IF TAPESTAT EQ 'E' STOP ENDIF  
  MOVE EMPLNUMB TO KEYAREA  
  MOVE EMPLDATA TO BDAMREC  
  WRITE DAFILE  
  IF BDAMSTAT NE 'Y' GOTO ERRORTN ENDIF  
  GOTO READLOOP  
.   
.
```

Figure 566 Create Variable-Length Records

In [Figure 566](#), the TAPEIN file contains records that are in employee number sequence. Each record contains an employee number (EMPLNUMB) and employee data (EMPLDATA). The program moves the number to the SEQUENTIAL data name as a key, moves the data to the BDAM file record area, and issues a WRITE for the BDAM file.

When the TAPEIN file reaches the end (STATUS dataname is equal to E), the program issues a STOP command. You always must specify when to end the run if a BDAM file is used.

Note: The first relative track number is 0.

Relative Track Number and Actual Record Number

Use VISION:Results statements to define the file to be created (output):

```
(VSE) FILE DAFILE BDAM OUTPUT FROM DAFILE F 35  
      SEQUENTIAL RELTRK RELADDR  
      LENGTH BLENGTH STATUS BDAMSTAT DISK 3350  
      EXTENTS 2 (optional)  
  
(z/OS) FILE DAFILE BDAM OUTPUT FROM DAFILE F 35  
        SEQUENTIAL RELTRK RELADDR  
        LENGTH BLENGTH STATUS BDAMSTAT
```

Figure 567 Sequential Create—Relative Track Number and Actual Record Number

Specify the record format (F, V, or U) and the record size.

The following keywords are required:

BDAM	Indicates BDAM file processing.
OUTPUT FROM filename	BDAM files can be output only from themselves.

SEQUENTIAL	No data name is required.
RELTRK dataname	Any name not defined elsewhere in your program. It holds the relative track number and the actual record number on that track. It is a 3-byte binary field.
LENGTH dataname	Required for undefined and variable-length records only. Any name not defined elsewhere in your program. It holds the length of the record to be written.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a WRITE has been issued for this file: Y Record was written correctly. N Space was not found in the file. L Length is incorrect. I Invalid request.
DISK type	VSE only. Defines the type of disk that contains the file. Valid values are 2311, 2314, 3330, 3340, and 3350.
EXTENTS n	VSE only. Number of EXTENT statements in the JCL for this file. Default is 1.

When records are written sequentially, you do not need to update the contents of the relative track address. VISION:Results updates it as follows:

- When a record is written, VISION:Results increments the actual record number by 1 and continues writing until space is no longer available on that track. VISION:Results increments the relative track number by 1, sets the actual record number to 1, and continues writing on the next track.
- The value of the relative track number and the actual record number used by the system are made available to you in the relative track address after the record is written.

To skip tracks, place the number of the track to advance to in the RELTRK dataname and reset the actual record number (probably to 1). For example, to write record 1 on track 4 before you have filled tracks 2 and 3, move X'000401' to the RELTRK dataname before issuing the WRITE.

Note: If you attempt to skip more records than are on a track or more tracks than are allocated, your program aborts.

To skip records within a track (fixed-length format only), put the next record number you want to write into the actual record number portion of the relative track address.

VISION:Results adds dummy records or writes capacity records to complete the intervening tracks or records.

If the initial relative track number value is not 0 (initial actual record value is not 1), VISION:Results completes the preceding tracks and/or records with dummy or capacity records.

If an attempt is made to write to an existing record or to write a record beyond the end of a track or outside of the file, an I is returned in the STATUS dataname.

If you attempt to write a record when end of file is reached, an N is returned in the STATUS dataname.

Undefined length records must be greater than or equal to 8 bytes; otherwise, an L is returned in the STATUS dataname.

To create variable-length records, see [Other Considerations on page 597](#).

The following procedure logic can be coded:

```
.  
.  
READLOOP:  
  READ TAPEIN  
  IF TAPESTAT EQ 'E' STOP ENDIF  
  MOVE EMPLDATA TO BDAMREC  
  WRITE DAFILE  
  IF BDAMSTAT NE 'Y' GOTO ERRORIN ENDIF  
  GOTO READLOOP  
.  
.
```

Figure 568 Coding the Procedure Logic

In [Figure 568](#), the TAPEIN file contains records with employee data. The program moves the data to the BDAM file record area and issues a WRITE to the BDAM file.

Note: The first relative track number is 0 and the first actual record number is 1.

When the TAPEIN file reaches the end (STATUS dataname is equal to E), the program issues a STOP command. You must always specify when to end the run if a BDAM file is used.

Relative Block Number

This addressing scheme can be used for fixed-length records only. It is not available for VSE.

Use VISION:Results statements to define the file to be created (output):

```
(z/OS) FILE DAFILE BDAM OUTPUT FROM DAFILE F 35  
      SEQUENTIAL RELBLK BLKNUMB STATUS BDAMSTAT
```

Figure 569 Sequential Create—Relative Block Number

Note: This is available in the z/OS version of VISION:Results only.

Specify the record format as F (fixed) and the record size.

The following keywords are required:

BDAM	Indicates BDAM file processing.
OUTPUT FROM filename	BDAM files can be output only from themselves.
SEQUENTIAL	No data name is required.
RELBLK dataname	Any name not defined elsewhere in your program. It holds the relative block number. It is a 3-byte binary field.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a WRITE has been issued for this file: Y Record was written correctly. N Space was not found in the file. I Invalid request.

When records are written sequentially, you do not need to update the contents of the relative block address. VISION:Results updates the contents as follows:

- Records are written on the first available track until space is no longer available.
- The value of the relative block number used by the system is made available to you in the relative block address after the record is written.

Note: If you attempt to skip more records than are on a track or more tracks than are allocated, the program aborts.

To skip blocks, place the number of the block to advance to in the RELBLK data name. For example, to write the thirtieth block, when the last relative block number is 10, move 30 to the RELBLK data name before issuing the WRITE.

Dummy records are added to complete the intervening blocks.

If the initial relative block number value is not 0, VISION:Results completes the preceding blocks with dummy records.

If you attempt to write an existing record or to write a record outside of the file, an I is returned in the STATUS dataname.

If you attempt to write a record when end of file is reached, an N is returned in the STATUS dataname.

The following procedure logic can be coded:

```
.  
.   
READLOOP:  
  READ TAPEIN  
  IF TAPESTAT EQ 'E' STOP ENDIF  
  MOVE EMPLDATA TO BDAMREC  
  WRITE DAFILE  
  IF BDAMSTAT NE 'Y' GOTO ERRORITN ENDIF  
  GOTO READLOOP  
.  
.
```

Figure 570 Coding the Procedure Logic

In [Figure 570](#), the TAPEIN file contains records with employee data. The program moves the data to the BDAM file record area and issues a WRITE to the BDAM file.

When the TAPEIN file reaches the end (STATUS dataname is equal to E), the program issues a STOP command. You must always specify when to end the run if a BDAM file is used.

The first relative block number is 0.

Random Read

This section describes reading records, one at a time and in no particular sequence, from a BDAM file. The program must identify precisely which record is wanted, prior to issuing the READ request. After the READ, one of two conditions exists: either the requested record is found, or it is not found.

To read a file, describe it to the operating system using JCL.

```
(VSE)      // ASSGN  SYSnnn,X'cuu'  
           // DLBL   filename,'DIRECT FILE',99365,DA  
           // EXTENT SYSnnn,...extent information  
  
(z/OS) //BDAMFILE DD DSN=BDAM.FILE.IDENT,DISP=SHR
```

Figure 571 Read a File Using JCL

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

The file name in your VISION:Results FILE statement must be the same as the file name or ddname specified in the previous JCL.

The following sections describe the statements used to read a BDAM file using the relative track number and actual key, relative track number and actual record number, or relative block number addressing scheme.

Relative Track Number and Actual Key

Use VISION:Results statements to define the input file:

```
(VSE)  FILE DAFILE  BDAM  RANDOM KEYAREA  F 35
        RELTRK TRKNUMB  KEYLEN 5
        STATUS BDAMSTAT  DISK 3350
        LENGTH BLENGTH  EXTENT 2 (optional keywords)

(z/OS) FILE DAFILE  BDAM  RANDOM KEYAREA  F 35
        RELTRK TRKNUMB  KEYLEN 5
        STATUS BDAMSTAT
        LENGTH BLENGTH (optional)
```

Figure 572 Random Read — Relative Track Number and Actual Key

Specify the record format (F, V, or U) and the record size.

The following keywords are required:

BDAM	Indicates BDAM file processing.
RANDOM dataname	Any name not defined elsewhere in your program. Its length is equal to the key length. It holds the key of the record to be retrieved. A character (CH) data type is assumed. If your key is signed numeric, packed decimal, or binary, code NU, PD, or BI, respectively, after dataname: RANDOM dataname PD Also, if the number of decimal places is other than zero, code it after the data type: dataname PD 2
RELTRK dataname	Any name not defined elsewhere in your program. It holds the relative track number. It is a 2-byte binary field.
KEYLEN n	Specifies the length of the record key. Allowable values for n are 1 to 255.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: Y Record was written correctly. N Space was not found in the file. I Invalid request.
DISK type	VSE only. Defines the type of disk that contains the file. Valid values are 2311, 2314, 3330, 3340, and 3350.
EXTENTS n	VSE only. Number of EXTENT statements in the JCL for this file. Default is 1.
LENGTH dataname	Any name not defined elsewhere in your program. It holds the length of the record obtained.

The following procedure logic can be coded:

```
.  
. .  
. .  
READLOOP:  
  READ TAPEIN  
  IF TAPESTAT EQ 'E' STOP ENDIF  
  MOVE EMPLNUMB TO KEYAREA  
  MOVE EMPLNUMB TO RAND WORK  
  PERFORM RANDOMIZE TO RAND_X  
  MOVE RAND WORK TO TRKNUMB  
  READ DAFILE  
  IF BDAMSTAT NE 'Y' GOTO NOTFOUND ENDIF  
  HEXPRINT BDAMREC  
  GOTO READLOOP  
. . .
```

Figure 573 Coding the Procedure Logic

Each record in a file TAPEIN contains an employee number (EMPLNUMB). The program moves the number to the RANDOM dataname as a key, randomizes the number and moves the result to the RELTRK as a relative track number, and issues a READ for the BDAM file. If a record is not read (STATUS dataname is not equal to Y), the program branches to an error routine. If the record is read correctly (STATUS dataname is equal to Y), it is printed and the program returns to read another TAPEIN record.

If an attempt is made to read a record outside of the file, an I is returned in the STATUS dataname.

The first byte of your key must not be high-values (X`FF'); otherwise, an I is returned in the STATUS dataname. If a record is not found, an N is returned in the STATUS dataname.

After a record is read, VISION:Results puts its length in the LENGTH data name.

When the TAPEIN file reaches the end (STATUS dataname is equal to E), the program issues a STOP command. You must always specify when to end the run if a BDAM file is used.

Note: The first relative track number is 0.

To read variable-length records, see [Other Considerations on page 597](#).

Relative Track Number and Actual Record Number

Use VISION:Results statements to define the input file:

```
(VSE)  FILE DAFILE BDAM RANDOM F 35
        RELTRK RELADDR STATUS BDAMSTAT
        DISK 3350
        LENGTH BLENGTH EXTENTS 2 (optional keywords)

(z/OS) FILE DAFILE BDAM RANDOM F 35
        RELTRK RELADDR STATUS BDAMSTAT
        LENGTH BLENGTH (optional)
```

Figure 574 Random Read—Relative Track Number and Actual Record Number

Specify the record format (F, V, or U) and the record size.

The following keywords are required:

BDAM	Indicates BDAM file processing.
RANDOM	No data name is required.
RELTRK dataname	Any name not defined elsewhere in your program. It holds the relative track number and the actual record number on that track. It is a 3-byte binary field.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: Y Record was written correctly. N Space was not found in the file. I Invalid request.
DISK type	VSE only. Defines the type of disk that contains the file. Valid values are 2311, 2314, 3330, 3340, and 3350.
EXTENTS n	VSE only. Number of EXTENT statements in the JCL for this file. Default is 1.
LENGTH dataname	Any name not defined elsewhere in your program. It holds the length of the record obtained.

The following procedure logic can be coded:

```
.
.
.
WORKAREA
ADDRESS 3 BI
REDEF AT ADDRESS
ADDR TT 2 BI
ADDR_R 1 BI
.
.
```

Figure 575 Coding the Procedure Logic (Page 1 of 2)

```
READLOOP:
  READ TAPEIN
  IF TAPESTAT EQ 'E' STOP ENDIF
  MOVE EMPLNUMB TO RAND WORK1
  PERFORM RANDOMIZE1 TO RAND X1
  MOVE RAND WORK1 TO ADDR TT
  MOVE EMPLNUMB TO RAND WORK2
  PERFORM RANDOMIZE2 TO RAND X2
  MOVE RAND WORK2 TO ADDR R
  MOVE ADDRESS TO RELADDR
  READ DAFILE
  IF BDAMSTAT NE 'Y' GOTO NOTFOUND ENDIF
  HEXPRINT BDAMREC
  GOTO READLOOP
.
```

Figure 575 Coding the Procedure Logic (Page 2 of 2)

Each record in a file TAPEIN contains an employee number (EMPLNUMB). The number is randomized and the result is moved to the RELTRK dataname as a track number, the number is randomized and moved to the RELTRK dataname as an actual record number on that track, and a READ is issued for the BDAM file.

If a record is not read (STATUS dataname is not equal to Y), the program branches to an error routine. If the record is read correctly (STATUS dataname is equal to Y), it is printed and the program returns to read another TAPEIN record.

After a record is read, VISION:Results puts its length in the LENGTH data name.

When the TAPEIN file reaches the end (STATUS dataname is equal to E), the program issues a STOP command. You must always specify when to end the run if a BDAM file is used.

If an attempt is made to read a record beyond the track or outside of the file, an I is returned in the STATUS dataname.

Note: The first relative track number is 0. The first actual record number is 1.

If a record is not found, an N is returned in the STATUS dataname.

To read variable-length records, see [Other Considerations on page 597](#).

Relative Block Number

Note: This section applies to z/OS only.

This addressing scheme can be used for fixed-length records only. It is not available for VSE.

Use VISION:Results statements to define the input file:

```
(z/OS) FILE DAFILE BDAM RANDOM F 35
      RELBLK BLKNUMB STATUS BDAMSTAT
```

Figure 576 Random Read—Relative Block Number (z/OS only)

Specify the record format as F (fixed) and the record size.

The following keywords are required:

BDAM	Indicates BDAM file processing.						
RANDOM	No data name is required.						
RELBLK dataname	The data name is any name not defined elsewhere in your program. It holds the relative block number. It is a 3-byte binary field.						
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: <table> <tr> <td>Y</td><td>Record was written correctly.</td></tr> <tr> <td>N</td><td>Space was not found in the file.</td></tr> <tr> <td>I</td><td>Invalid request.</td></tr> </table>	Y	Record was written correctly.	N	Space was not found in the file.	I	Invalid request.
Y	Record was written correctly.						
N	Space was not found in the file.						
I	Invalid request.						

The following procedure logic can be coded:

```
.
.
.
READLOOP:
  READ TAPEIN
  IF TAPESTAT EQ 'E' STOP ENDIF
  MOVE EMPLNUMB TO RAND WORK
  PERFORM RANDOMIZE TO RAND_X
  MOVE RAND WORK TO BLKNUMB
  READ DAFILE
  IF BDAMSTAT NE 'Y' GOTO NOTFOUND ENDIF
  HEXPRINT BDAMREC
  GOTO READLOOP
.
.
.
```

Figure 577 Coding the Procedure Logic

In [Figure 577](#), each record in a file TAPEIN contains an employee number (EMPLNUMB). The program randomizes the number, moves the result to the RELBLK data name as a relative block number, and issues a READ for the BDAM file. If the record is not read (STATUS dataname is not equal to Y), the program branches to an error routine. If the record is read correctly (STATUS dataname is equal to Y), it is printed and the program returns to read another TAPEIN record.

If an attempt is made to read a record outside of the file, an I is returned in the STATUS dataname.

If a record is not found, an N is returned in the STATUS dataname.

Note: The first relative block number is 0.

When the TAPEIN file reaches the end (STATUS dataname is equal to E), a STOP command is issued. It is always necessary to tell VISION:Results when to end the run if a BDAM file is used.

Sequential Read

This section describes reading a BDAM file starting with the first record and reading serially until end of file occurs or a STOP command is issued by the VISION:Results program.

To read a file, describe it to the operating system using JCL.

VSE

```
// ASSGN  SYSnnn,X'cuu'  
// DLBL   filename,'DIRECT FILE',99365,DA  
// EXTENT SYSnnn,...extent information
```

Figure 578 BDAM File Sequential Read (VSE)

z/OS

```
//BDAMFILE DD DSN=BDAM.FILE.IDENT,DISP=SHR
```

Figure 579 BDAM File Sequential Read (z/OS)

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

The file name in your FILE statement must be the same as the file name or ddname specified in the previous JCL.

When a BDAM file is being read sequentially, relative addresses RELTRK dataname and RELBLK dataname are not updated.

Note: The data name is required with the RELTRK and RELBLK keywords.

All records are returned to you, including dummy records (z/OS fixed-length records only). Records cannot be skipped.

For information about reading a BDAM file using relative track number and actual key, relative track number and actual record number, or relative block number addressing scheme, see [Sequential Read on page 586](#).

Relative Track Number and Actual Key

Use VISION:Results statements to define the input file:

```
(VSE)  FILE DAFILE BDAM SEQUENTIAL KEYAREA F 35
        RELTRK TRKNUMB KEYLEN 5
        STATUS BDAMSTAT DISK 3350
        LENGTH BLENGTH EXTENTS 2 (optional keywords)

(z/OS) FILE DAFILE BDAM SEQUENTIAL KEYAREA F 35
        RELTRK TRKNUMB KEYLEN 5
        STATUS BDAMSTAT
        LENGTH BLENGTH (optional)
```

Figure 580 Sequential Read — Relative Track Number and Actual Key

Specify the record format (F, V, or U) and the record size.

The following keywords are required:

BDAM	Indicates BDAM file processing.
SEQUENTIAL dataname	The dataname is not required. It is any name not defined elsewhere in your program. Its length is equal to the key length. It holds the key of the record obtained. A character (CH) data type is assumed. If your key is signed numeric, packed decimal, or binary, code NU, PD, or BI, respectively, after dataname: SEQUENTIAL dataname PD Also, if the number of decimal places is other than zero, code it after the data type: SEQUENTIAL dataname PD 2
RELTRK dataname	Any name not defined elsewhere in your program.
KEYLEN n	Required only if SEQUENTIAL dataname is used. Specifies the length of the record key. Allowable values for n are 1 to 255.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: Y Record was read correctly. D Dummy record (z/OS fixed-length records only). E End of file.
DISK type	VSE only. Defines the type of disk that contains the file. Valid values are 2311, 2314, 3330, 3340, and 3350.
EXTENTS n	VSE only. Number of EXTENT statements in the JCL for this file. Default is 1.
LENGTH dataname	Any name not defined elsewhere in your program. It holds the length of the record obtained.

The following procedure logic can be coded:

```
.  
.
READLOOP:
  READ DAFILE
  IF BDAMSTAT EQ 'E' STOP ENDIF
  IF BDAMSTAT EQ 'D'          } z/OS only
    GOTO READLOOP ENDIF      } z/OS only
  HEXPRINT KEYAREA
  HEXPRINT BDAMREC
  GOTO READLOOP
.  
.
```

Figure 581 Coding the Procedure Logic

If an actual record is read (STATUS dataname is equal to Y), it is printed. If a dummy record is read (STATUS dataname is equal to D), the program reads the next one (z/OS only). When the DAFILE reaches the end (STATUS dataname is equal to E), the program issues a STOP command. You must always specify when to end the run if a BDAM file is used.

The data name does not need to be specified in the SEQUENTIAL keyword. If a data name is specified, the record's key is placed into that field.

To read variable-length records, see [Other Considerations on page 597](#).

Relative Track Number and Actual Record Number

Use VISION:Results statements to define the input file:

```
(VSE)  FILE DAFILE BDAM SEQUENTIAL F 35
        RELTRK RELADDR STATUS BDAMSTAT DISK 3350
        LENGTH BLENGTH EXTENTS 2 (optional keywords)

(z/OS) FILE DAFILE BDAM SEQUENTIAL F 35
        RELTRK RELADDR STATUS BDAMSTAT
        LENGTH BLENGTH (optional)
```

Figure 582 Sequential Read—Relative Track Number and Actual Record Number

Specify the record format (F, V, or U) and the record size.

The following keywords are required:

BDAM	Indicates BDAM file processing.
SEQUENTIAL	No data name is required.
RELTRK dataname	Any name not defined elsewhere in your program.

STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: Y Record was read correctly. D Dummy record (z/OS fixed-length records only). E End of file.
LENGTH dataname	Any name not defined elsewhere in your program. It holds the length of the record obtained.
DISK type	VSE only. Defines the type of disk that contains the file. Valid values are 2311, 2314, 3330, 3340, and 3350.
EXTENTS n	VSE only. Number of EXTENT statements in the JCL for this file. Default is 1.

The following procedure logic can be coded:

```

.
.
READLOOP:
  READ DAFILE
  IF BDAMSTAT EQ 'E' STOP ENDIF
  IF BDAMSTAT EQ 'D'          } z/OS only
    GOTO READLOOP ENDIF      } z/OS only
  HEXPRINT BDAMREC
  GOTO READLOOP
.
.

```

Figure 583 Coding the Procedure Logic

If an actual record is read (STATUS dataname is equal to Y), it is printed. If a dummy record is read (STATUS dataname is equal to D), the program reads the next one (z/OS only). When the DAFILE reaches the end (STATUS dataname is equal to E), the program issues a STOP command. You must always specify when to end the run if a BDAM file is used.

To read variable-length records, see [Other Considerations on page 597](#).

Relative Block Number

Note: This section applies to z/OS only.

This addressing scheme can be used for fixed-length records only. It is not available for VSE. Use VISION:Results statements to define the input file:

```

(z/OS)  FILE DAFILE BDAM SEQUENTIAL F 35
        RELBLK BLKNUMB STATUS BDAMSTAT

```

Figure 584 Sequential Read—Relative Block Number (z/OS only)

Specify the record format as F (fixed) and the record size.

The following keywords are required:

BDAM	Indicates BDAM file processing.
SEQUENTIAL	No data name is required.
RELBLK dataname	Any name not defined elsewhere in your program.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values after a READ has been issued for this file: Y Record was read correctly. D Dummy record (z/OS fixed-length records only). E End of file.

The following procedure logic can be coded:

```
.  
.  
READLOOP:  
  READ DAFILE  
  IF BDAMSTAT EQ 'E' STOP ENDIF  
  IF BDAMSTAT EQ 'D' GOTO READLOOP ENDIF  
  HEXPRINT BDAMREC  
  GOTO READLOOP  
.  
.  
.
```

Figure 585 Coding the Procedure Logic

If an actual record is read (STATUS dataname is equal to Y), it is printed. If a dummy record is read (STATUS dataname is equal to D), the program reads the next one (z/OS only). When the DAFILE reaches the end (STATUS dataname is equal to E), the program issues a STOP command. You must always specify when to end the run if a BDAM file is used.

Random Update

This section describes the random insertion and changing of records on a BDAM file. The file is updated in place, without copying the old file to a new file.

To update a file, describe the file to the operating system using JCL.

VSE

```
// ASSGN SYSnnn,X'cuu'  
// DLBL  filename,'DIRECT FILE',99365,DA  
// EXTENT SYSnnn,...extent information
```

Figure 586 BDAM File Random Update (VSE)

Note: This type of run is not restartable. Once one record has been added or changed, the file is not the same, regardless of why the job ended. Create a method of recovering the original file as a precaution before running a program that uses this technique.

z/OS

```
//BDAMFILE DD DSN=BDAM.FILE.IDENT,DISP=OLD
```

Figure 587 BDAM File Random Update (z/OS)

In [Figure 587](#), note the disposition of OLD.

The particulars of the JCL required vary from one installation to another. To code the JCL successfully, you must be familiar with your installation standards.

The file name in your VISION:Results FILE statement must be the same as the file name or ddname specified in the previous JCL.

If you do not issue a READ for a given record before you issue the WRITE, your program aborts.

Note: Before a record can be replaced, it must be read and *found*. Before a record can be added, it must be read and *not found*, even if you know it is not there.

For the statements needed to randomly update a BDAM file using relative track number and actual key, relative track number and actual record number, or the relative block number addressing scheme, see [Random Update on page 590](#).

Relative Track Number and Actual Key

Use VISION:Results statements to define the input/output file:

```
(VSE)  FILE DAFILE BDAM IO RANDOM KEYAREA F 35
        RELTRK TRKNUMB KEYLEN 5
        STATUS BDAMSTAT DISK 3350
        LENGTH BLENGTH EXTENTS 2 (optional keywords)

(z/OS) FILE DAFILE BDAM IO RANDOM KEYAREA F 35
        RELTRK TRKNUMB KEYLEN 5
        STATUS BDAMSTAT
        LENGTH BLENGTH (optional)
```

Figure 588 Random Update—Relative Track Number and Actual Key

Specify the record format (F, V, or U) and the record size.

The following keywords are required:

BDAM	Indicates BDAM file processing.
IO	Allows the file to be updated in place. Record format and record size must be specified on the FILE statement in conjunction with this keyword.

RANDOM dataname Any name not defined elsewhere in your program. Its length is equal to the key length. It holds the key of the record to be retrieved. A character (CH) data type is assumed. If your key is signed numeric, packed decimal, or binary, code NU, PD, or BI, respectively, after dataname:

RANDOM dataname PD

Also, if the number of decimal places is other than zero, code it after the data type:

RANDOM dataname PD 2

RELTRK dataname Any name not defined elsewhere in your program. It holds the relative track number. It is a 2-byte binary field.

KEYLEN n Specifies the length of the record key. Allowable values for n are 1 to 255.

STATUS dataname Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values:

After a READ is issued:

Y Record was found and read correctly.
N Record was not found.
I Invalid request.

After a WRITE is issued:

Y Record was written correctly.
N Space was not found on the track.
L Length is incorrect.

DISK type VSE only. Defines the type of disk that contains the file. Valid values are 2311, 2314, 3330, 3340, and 3350.

EXTENTS n VSE only. Number of EXTENT statements in the JCL for this file. Default is 1.

LENGTH dataname Required only for undefined and variable-length records. Any name not defined elsewhere in your program. It holds the length of the record.

If an attempt is made to read a record outside of your file, an I is returned in the STATUS dataname.

The first byte of your key must not be high-values (X'FF'); otherwise, an I is returned in the STATUS dataname.

If the relative track number, the actual key, or the record length (if the record was found) is changed before the WRITE is issued, your program aborts.

Undefined length records must be greater than or equal to 8 bytes; otherwise, an L is returned in the STATUS dataname.

The following procedure logic can be coded:

```
.
.
.
READLOOP:
  READ TAPEIN
  IF TAPESTAT EQ 'E' STOP ENDIF
  MOVE EMPLNUMB TO KEYAREA
  MOVE EMPLNUMB TO RAND WORK
  PERFORM RANDOMIZE TO RAND_X
  MOVE RAND WORK TO TRKNUMB
  READ DAFILE
  IF BDAMSTAT NE 'Y' GOTO ADDRRECD ENDIF
  REWRRECD:
    HEXPRINT BDAMREC
    IF EMPHONE NE BLANKS MOVE EMPHONE TO BDPHONE ENDIF
    IF EMPADDR NE BLANKS MOVE EMPADDR TO BDADDR ENDIF
    WRITE DAFILE
    IF BDAMSTAT NE 'Y' GOTO ERWRITE ENDIF
    GOTO READLOOP
  ADDRRECD:
    IF BDAMSTAT NE 'N' GOTO ERREAD ENDIF
    MOVE EMPLDATA TO BDAMREC
    WRITE DAFILE
    IF BDAMSTAT NE 'Y' GOTO ERWRITE ENDIF
    GOTO READLOOP
.
.
.
```

Figure 589 Coding the Procedure Logic

In [Figure 589](#), each record in a file TAPEIN contains an employee number (EMPLNUMB) and employee data (EMPLDATA). The program moves the number to RANDOM dataname as a key, randomizes the number and moves the result to the RELTRK dataname as a relative track number, and issues a READ for the BDAM file. If the record is found (STATUS dataname is equal to Y), the program changes the fields to be updated and replaces the record. If the record is not found (STATUS dataname is equal to N), the program adds it to the file.

When the TAPEIN file reaches the end (STATUS dataname is equal to E), the program issues a STOP command. You must always specify when to end the run if a BDAM file is used.

Note: The first relative track number is 0.

To update variable-length records, see [Considerations for Processing Variable-Length Records on page 136](#).

Relative Track Number and Actual Record Number

You can replace fixed, variable, and undefined length records. You can add only fixed-length records.

Use VISION:Results statements to define the input/output file:

```
(VSE)  FILE DAFILE BDAM IO RANDOM F 35
      RELTRK RELADDR STATUS BDAMSTAT
      DISK 3350
      LENGTH BLENGTH EXTENTS 2 (optional keywords)

(z/OS) FILE DAFILE BDAM IO RANDOM F 35
      RELTRK RELADDR STATUS BDAMSTAT
      LENGTH BLENGTH (optional)
```

Figure 590 Random Update—Relative Track Number and Actual Record Number

Specify the record format (F, V, or U) and the record size.

The following keywords are required:

BDAM	Indicates BDAM file processing.
IO	Allows the file to be updated in place. Record format and record size must be specified on the FILE statement in conjunction with this keyword.
RANDOM	No data name is required.
RELTRK dataname	Any name not defined elsewhere in your program. It holds the relative track number and the actual record number on that track. It is a 3-byte binary field.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values:

After a READ is issued:

Y	Record was found.
N	Record was not found.
I	Invalid request.

After a WRITE is issued:

Y	Record was written correctly.
L	Length is incorrect.

DISK type	VSE only. Defines the type of disk that contains the file. Valid values are 2311, 2314, 3330, 3340, and 3350.
EXTENTS n	VSE only. Number of EXTENT statements in the JCL for this file. Default is 1.
LENGTH dataname	Required only for undefined and variable-length records. Any name not defined elsewhere in your program. It holds the length of the record.

If an attempt is made to read a record beyond the end of the track or outside of your file, an I is returned in the STATUS dataname.

If the relative track number, the actual record number, or the record length (if the record was found), is changed before the WRITE is issued, your program aborts.

Undefined length records must be greater than or equal to 8 bytes; otherwise, an L is returned in the STATUS dataname.

To update variable-length records, see [Other Considerations on page 597](#).

The following procedure logic can be coded:

```
.
.
WORKAREA
  ADDRESS 3 BI
  REDEF AT ADDRESS
  ADDR TT 2 BI
  ADDR R 1 BI
.
.
READLOOP:
  READ TAPEIN
  IF TAPESTAT EQ 'E' STOP ENDIF
  MOVE EMPLNUMB TO RAND WORK1
  PERFORM RANDOMIZE1 TO RAND X1
  MOVE RAND WORK1 TO ADDR TT
  MOVE EMPLNUMB TO RAND WORK2
  PERFORM RANDOMIZE2 TO RAND X2
  MOVE RAND WORK2 TO ADDR R
  MOVE ADDRESS TO RELADDR
  READ DAFILE
  IF BDAMSTAT NE 'Y' GOTO ADDRRECD ENDIF
  REWRRECD:
    HEXPRINT BDAMREC
    IF EMPHONE NE BLANKS MOVE EMPHONE TO BDPHONE ENDIF
    IF EMPADDR NE BLANKS MOVE EMPADDR TO BDADDR ENDIF
    WRITE DAFILE
    IF BDAMSTAT NE 'Y' GOTO ERWRITE ENDIF
    GOTO READLOOP
  ADDRRECD:
    IF BDAMSTAT NE 'N' GOTO ERREAD ENDIF
    MOVE EMPLDATA TO BDAMREC
    WRITE DAFILE
    IF BDAMSTAT NE 'Y' GOTO ERWRITE ENDIF
    GOTO READLOOP
.
.
```

Figure 591 Coding the Procedure Logic

Each record in a file TAPEIN contains an employee number (EMPLNUMB) and employee data (EMPLDATA). The program randomizes the number and moves the result to the RELTRK dataname as a relative track number, randomizes the number and moves the result to the RELTRK dataname as an actual record number on that track, and issues a READ for the BDAM file. If the record is found (STATUS dataname is equal to Y), the program changes the fields to be updated and replaces the record. If the record is not found (STATUS dataname is equal to N), the program adds it to the file.

When the TAPEIN file reaches the end (STATUS dataname is equal to E), the program issues a STOP command. You must always specify when to end the run if a BDAM file is used.

Note: The first relative track number is 0 and the first actual record number is 1.

Relative Block Number

Note: This section applies to z/OS only.

This addressing scheme can be used for fixed-length records only. It is not available for VSE.

Use VISION:Results statements to define the input/output file:

```
(z/OS) FILE DAFILE BDAM IO RANDOM F 35  
      RELBLK BLKNUMB STATUS BDAMSTAT
```

Figure 592 Random Update—Relative Block Number (z/OS only)

Specify a record format of F (fixed) and the record size.

The following keywords are required:

BDAM	Indicates BDAM file processing.
IO	Allows the file to be updated in place. Record format and record size must be specified on the FILE statement in conjunction with this keyword.
RANDOM	No data name is required.
RELBLK dataname	Any name not defined elsewhere in your program. It holds the relative block number. It is a 3-byte binary field.
STATUS dataname	Any name not defined elsewhere in your program. It is a 1-byte field that contains one of the following values:

After a READ is issued:

Y	Record was found and read correctly.
N	Record was not found.
I	Invalid request.

After a WRITE is issued:

Y	Record was written correctly.

If an attempt is made to read a record outside of your file, an I is returned in the STATUS dataname.

If the relative block number is changed before a WRITE is issued, your program aborts.

The following procedure logic can be coded:

```
.
.
READLOOP:
  READ TAPEIN
  IF TAPESTAT EQ 'E' STOP ENDIF
  MOVE EMPLNUMB TO RAND WORK
  PERFORM RANDOMIZE TO RAND_X
  MOVE RAND WORK TO BLKNUMB
  READ DAFILE
  IF BDAMSTAT NE 'Y' GOTO ADDRCDR ENDIF
ADDRCDR:
  HEXPRINT BDAMREC
  IF BDAMSTAT NE 'Y' GOTO ERWRITE ENDIF
  IF EMPHONE NE BLANKS MOVE EMPHONE TO BDPHONE ENDIF
  IF EMPADDR NE BLANKS MOVE EMPADDR TO BDADDR ENDIF
  WRITE DAFILE
  GOTO READLOOP
ADDRCDR:
  IF BDAMSTAT NE 'N' GOTO ERREAD ENDIF
  MOVE EMPLDATA TO BDAMREC
  WRITE DAFILE
  IF BDAMSTAT NE 'Y' GOTO ERWRITE ENDIF
  GOTO READLOOP
.
```

Figure 593 Coding the Procedure Logic

Each record in a file TAPEIN contains an employee number (EMPLNUMB) and employee data (EMPLDATA). The program randomizes the number, moves the result to the RELBLK data name as a relative block number, and issues a READ for the BDAM file. If the record is found (STATUS dataname is equal to Y), the program changes the fields to be updated and replaces the record. If the record is not found (STATUS dataname is equal to N), the program adds it to the file.

When the TAPEIN file reaches the end (STATUS dataname is equal to E), the program issues a STOP command. You must always specify when to end the run if a BDAM file is used.

Note: The first relative block number is 0.

Other Considerations

This section describes other BDAM processing considerations.

Processing Variable-Length Records

VISION:Results simplifies the processing of files with variable-length records.

Record Size	When specifying the logical record length for a variable file, specify the maximum data length (not including the length of the RDW).
Block Size	Blocking has no meaning for BDAM files and is ignored.

Location of Data within Record	VISION:Results assumes that position 1 of every variable record is the first byte of data. VISION:Results automatically takes care of the 4-byte RDW for BDAM variable files. When setting up data definitions for variable-length records, start defining fields from the first byte of data.
Length of Current Record	On input, VISION:Results keeps track of the length of the data in the record just read. You can access this information by specifying the keyword LENGTH and a data name. The data name specified this way is a 2-byte binary field and contains the length of the data in the record just read. For output files, it is your responsibility to tell VISION:Results how long the data portion of the current record is.
Output from Itself	If a file is defined to be output from itself, it has its own I/O area assigned. It also has a 2-byte binary length field assigned to the file. You can reference this length field by specifying the keyword LENGTH followed by a data name. It is your responsibility to move data to the area and place the length of the data in the LENGTH data name before the record is to be written, either using the automatic cycle or a WRITE command. Failure to move a valid length to the LENGTH data name can result in wrong length records being written.
Output from an Input File	This is not available.
File Printing Variable Files	When a variable-length BDAM file is file printed (HEXPRINT, PRINT, and so on), the RDW and the data portion of the record are printed. The maximum length of the record is always printed, regardless of the contents of the RDW. The RDW does not have the correct value in it until after a WRITE occurs. Either print the record after a WRITE or print the LENGTH data name in conjunction with printing the record.
Sorting a Variable File	To eliminate errors during SORT, variable-length records are moved from the I/O area to a fixed-length area for processing. This is transparent to you, and you do not need to be concerned with this. If an error occurs in the VISION:Results execution, the record shown in the I/O area in the PRINTERERROR dump is not necessarily the record you are processing. The variable record has been moved to a work area. Check your record there.
Record Format Specifications	z/OS users must specify variable record format in their FILE statement to have VISION:Results handle the RDW automatically. If the record format is not specified at compile time, VISION:Results passes you the ENTIRE record (including the RDW) in the record area.

Numeric and Packed Key

When processing a BDAM file where you are building or retrieving records, consider the sign of the key if the key has a data type of numeric (zoned decimal) or packed. A positive sign for a numeric or packed field can be C or F. This can be a problem if you are searching for a particular record on a BDAM file, and the key you are using to search has a different sign than the key in the record. For example, a 2-byte packed field with the value 123C is not considered equal to a 2-byte packed field with the value 123F. A NO RECORD FOUND status is set.

In a VISION:Results program, you can run into an unlike sign problem under the following circumstances: you have set up a packed key in the BDAM key area and this key has developed a C sign somewhere along the line (perhaps the key had been converted from numeric to packed and a C sign resulted from this process). If the key in the record contains an F sign, a status code of N (no record found) is returned to the program after a random retrieval read. The same situation can occur with numeric fields.

To prevent or correct for unlike signs, use one of the following approaches.

Whenever possible, treat the keys as character data. In [Figure 595](#), the keys have a data type of packed but are defined as character, so no sign change can occur. If the key has to be treated as packed, it can be redefined.

```
FILE BDAMIN2 F 130 RANDOM FINDKEY RELTRK RELADDR2
  STATUS BDSTAT KEYLEN 5
  ACCOUNT 5 NU ...
FILE TRANSIN STATUS TRANSTAT
  TRANACCT 5 ...
  NUMBIN2 3 BI
PROCEDURE:
  IF TRANSTAT EQ 'E' STOP ENDIF
  MOVE TRANACCT TO FINDKEY
  MOVE NUMBIN2 TO RELADDR2
  READ BDAMIN2
.
```

Figure 594 Numeric Key is Defined as Character

In [Figure 594](#), a numeric key in the transaction file (TRANSIN) is defined as character. FINDKEY is also defined as character (default). No sign change occurs when TRANACCT is moved to FINDKEY.

```
FILE BDAMIN F 120 RANDOM SEARCHKEY RELTRK RELADDR
  STATUS BDAMSTAT KEYLEN 3
FILE TRANIN STATUS TRANEOF
  KEYIN 3 CH ; (key is actually packed decimal)
  NUMBIN 3 BI
PROCEDURE:
  IF TRANEOF EQ 'E' STOP ENDIF
  MOVE KEYIN TO SEARCHKEY ; (sign remains unchanged)
  MOVE NUMBIN TO RELADDR
  READ BDAMIN
  IF BDAMSTAT NE 'Y' GOTO NOTFOUND ENDIF
.
```

Figure 595 Treat the Keys as Character Data

To convert a packed key's sign from C to F, use OR on the field. In [Figure 596](#), a 2-byte packed key is having its sign set to F.

```
COMBINE BITS X'000F' OR SRCHKEY
```

Figure 596 Convert a Packed Key's Sign from C to F

To convert a packed key's sign from F to C, use AND on the field. In [Figure 597](#), a 2-byte packed key with an F sign is having its sign set to C.

```
COMBINE BITS X'FFFC' AND SRCHKEY
```

Figure 597 Convert a Packed Key's Sign from F to C

BDAM Examples

The VISION:Results test data included on your installation tape includes the following BDAM examples.

Example 1 Random Create, Fixed-Length Record with a Key

```
OPTION STRUCTURED
FILE ARFILE FB 352 5280 STATUS ASTAT
  ACCTREC 352
  ACCTNO    7 4 NU

FILE DACCTS BDAM F 352 OUTPUT FROM DACCTS RANDOM DAKEY NU
  KEYLEN 7 RELTRK DATT STATUS DASTAT COUNT DACNT

  DACCTREC 352

WORKAREA
  PRIME      1 BI VALUE 29 ; 30 TRACKS ALLOCATED TO FILE
  QUOTIENT   4 BI
  REMAINDER  1 BI
READ ARFILE
DOWHILE ASTAT NE 'E' ; READ RECORDS UNTIL END OF FILE
  QUOTIENT = ACCTNO / PRIME ; USE PRIME
  REMAINDER = ACCTNO - (QUOTIENT * PRIME) ; NUMBER DIVISION
  MOVE REMAINDER TO DATT ; REMAINDER
  MOVE ACCTNO TO DAKEY ; TECHNIQUE
  MOVE ACCTREC TO DACCTREC ; TO CALCULATE
  WRITE DACCTS ; TRACK NUMBER
  IF DASTAT EQ 'N' DATT = DATT + 1 ; ALLOW OVERFLOW TO NEXT
  WRITE DACCTS ; TRACK ONLY
  IF DASTAT EQ 'N' LIST ACCTNO 'FILE IS FULL' DATT
  MOVE 400 TO DYLRRETURN
  EXITDO
ENDIF
  DYLCOUNT2 = DYLCOUNT2 + 1
ENDIF
IF DASTAT EQ 'I'
  LIST ACCTNO 'RANDOMIZING ERROR' DATT
```

Figure 598 [Example 1](#) Random Create, Fixed-Length Record with a Key
(Page 1 of 2)


```

        MOVE 401 TO DYLRETURN
        EXITDO
    ENDIF
    READ ARFILE
ENDDO
MOVE 'DACCTS' TO DYLCNAME1
MOVE DACNT TO DYLCOUNT1
MOVE 'OVERFLOW' TO DYLCNAME2
STOP

```

Figure 598 [Example 1](#) Random Create, Fixed-Length Record with a Key
(Page 2 of 2)

Control Totals of Example 1

VISION:RESULTS CONTROL TOTALS				
FILE ID	RECORD COUNT	CHARACTER COUNT	BLOCK COUNT	DROPPED BLOCK COUNT
ARFILE	200	70,400		
		RECORDS		PAGES
FILE PRINT				1
REPORT PRINT				
FIXED BLANK COUNT				
FIXED DECIMAL DIVIDE				
DACCTS				200
OVERFLOW				

Figure 599 [Example 1](#) Totals of Random Create, Fixed-Length Record with a Key

Automatic control totals do not occur for a BDAM file; therefore, the DYLCOUNT fields are used instead.

If space cannot be found to add a record to a track or its following overflow track, either more space must be allocated to the file and a new prime number used, or a different randomizing algorithm must be used.

Example 2 Sequential Read, Fixed-Length Record with a Key

```

OPTION STRUCTURED

FILE DACCTS BDAM F 352 SEQUENTIAL
  KEYLEN 7 RELTRK DATT STATUS DASTAT COUNT DACNT
DELFLAG 1      ; X'FF' MEANS RECORD WAS LOGICALLY DELETED
ACCTNO  7      4 NU (ACCOUNT NUMBER)
NAME     25    85 (NAME)
ADD1     25   110 (ADDRESS LINE 1)
ADD2     25   135 (CITY, STATE AND ZIP)

DUNTIL DASTAT EQ 'E' OR      ; READ RECORDS UNTIL END OF FILE
(DASTAT NE 'D' AND          ; BYPASSING DUMMY RECORDS AND
DELFLAG NE X'FF')           ; LOGICALLY DELETED RECORDS
READ DACCTS
ENDDO
SORT DACCTS USING ACCTNO
LIST ACCTNO NAME ADD1 ADD2

ON END OF SORTING
MOVE 'DACCTS' TO DYLCNAME1
MOVE DACNT TO DYLCOUNT1
STOP

T1 'ACCOUNTS DIRECTORY'
T1 + 1 DYLDATE
T1 + 100 DYLPAGE

```

Figure 600 [Example 2](#) Sequential Read, Fixed-Length Record with a Key**Control Totals of Example 2**

VISION:RESULTS CONTROL TOTALS				
FILE ID	RECORD COUNT	CHARACTER COUNT	BLOCK COUNT	DROPPED BLOCK COUNT
		RECORDS		PAGES
FILE PRINT				1
REPORT PRINT				4
FIXED BLANK COUNT				
FIXED DECIMAL DIVIDE				
DACCTS			200	

Figure 601 [Example 2](#) Totals of Sequential Read, Fixed-Length Record with a Key

Output of Example 2

01/15/99	ACCOUNT NUMBER	NAME	ACCOUNTS DIRECTORY ADDRESS LINE 1	PAGE CITY, STATE AND ZIP
	0000434		573 INEZ	LOS ANGELES CA 90023
	0002623		573 SO BOYLE AVE	LOS ANGELES CA 90017
	1001191	CARDOZA, MARY	1008 WESTMORELAND	LA CA90007
	1002775	S FE EMP HOSP ASSN	1063 PACIFIC AVE	LA CA90023
	1006231	DE LAUNEY, RAYMOND	1432 LOMITA	ARCADIA CA91006
	1006681	MYERS, EARL	3735 84TH	LA CANADA CA91011
	1009681		305 SO BOYLE AVE	INGLEWOOD CA
	1013637	STALLINS, LEO F	1655 CHAPEL	RIALTO CA92376
	1013904	HINDERLICH, PAUL	2835 SANTA INEZ	LA CA90032
	1014021	WOOD, ROGER D	123 MONTARA	PINEDALE CA93650
	1014617	MORRIS, CLARENCE E	214 E 55TH	LONG BEACH CA90802
	1014706	HOLLENBECK HOME TR	3734 LANERANCO STREE	LA CA90033
	1014889	SULLIVAN, JOSEPH C	635 GUIRADO ST	BELLFLOWER CA90706
	2000504	WILSON,HERNMAN	1946 W 43RD PL	LOS ANGELES CA 90051
	2002299	PLACIDO,ORTEGA	1247 S ST LOUIS ST	LOS ANGELES CA 90023
	2002922	FLOWERS,ETHEL	4420 ARKANSAS	LOS ANGELES CA 90048
	2004739	STEIDLEY,GUTHA N	330 S ST LOUIS ST	SAN BERNADINO CA 92410
	2005131	WICINSKI,ALEXANDER B	610 SOLWAY ST	SOUTHGATE CA 90280
	2005956	SOTO,RUFINO	1454 W BL AP B	ONTARIO CA
	2006286	BALDWIN,HARRY R	7015 NO VAN WIG AVE	NEEDLES CA 93303
	2007177	HART,GRADDIE	513 CIMARRON ST	HIGHLAND CA 92346
	2008831	ZERING,WILLIAM G	4723 E 37TH ST	VAN NUYS CA
	2009072	JONES,RUFUS G	3131 COLLIS AVE	LOS ANGELES CA
	2010933	QUINTINAR,ANTONIO G	3934 HOBART	LA CA 90063
	2011719	REED,LOWELL D	3255 W 77TH ST	LOGANSFORT INDIANA
	2013568	CULLENDER,EVERRETT J	1503 GEORGE ST APR #	FONTANA CA 92335
	2014017	HERNANDEZ,CARLOS	3171 E AVE 28	LOS ANGELES CA 90012
	4003896	PETERSON, WALTER	610 N GARDNER AVE	CULVER CITY CA90230
	6001327	CARLON,MR MARIANO	3417 CEDAR	LOS ANGELES CA

Figure 602 [Example 2](#) Output of Sequential Read, Fixed-Length Record with a Key

In VSE, dummy records are not returned to the program; therefore, a check need not be made for them.

It is not possible to physically delete a record from a BDAM file. Deleting can be accomplished only logically. A field in the record (usually the first byte) is selected to contain a flag (usually X`FF'). When the flag is on the record, the record is treated as though it did not exist.

Example 3 Random Update, Fixed-Length Record with a Key

```
OPTION STRUCTURED

REPORT01 ; BACK-UP BDAM FILE BEFORE UPDATE IN PLACE

FILE DACCTS BDAM F 352 SEQUENTIAL
  KEYLEN 7 RELTRK DATT STATUS DASTAT COUNT DACNT
DACCTREC 352
DELFLAG 1 1 ; X'FF' MEANS RECORD HAS BEEN LOGICALLY DELETED

FILE TACCTS FB 352 OUTPUT FROM TACCTS
TACCTREC 352

READ DACCTS
DOWHILE DASTAT NE 'E' ; READ RECORDS UNTIL END OF FILE
  IF DASTAT NE 'D' AND ; BYPASS DUMMY RECORDS AND
    DELFLAG NE X'FF' ; LOGICALLY DELETED RECORDS
    MOVE DACCTREC TO TACCTREC
  WRITE TACCTS
ENDIF
READ DACCTS
ENDDO
MOVE 'DACCTS' TO DYLCNAME1
MOVE DACNT TO DYLCOUNT1
STOP
```

Figure 603 [Example 3](#) Random Update, Fixed-Length Record with a Key**Control Totals of Example 3**

```
VISION:RESULTS CONTROL TOTALS

FILE      RECORD      CHARACTER      BLOCK      DROPPED
ID         COUNT       COUNT        COUNT      BLOCK COUNT

ARFILE           200           70,400

                                RECORDS                PAGES

FILE PRINT                                           1
REPORT PRINT                                          1
FIXED BLANK COUNT
FIXED DECIMAL DIVIDE
DACCTS                                           200
```

Figure 604 [Example 3](#) Totals of Random Update, Fixed-Length Record with a Key

Because a BDAM update is not restartable, as a precaution against loss of the file, a copy is made of the file before the update occurs. If the update aborts, then the BDAM file could be restored using the program shown in [Example 1](#).

In VSE, dummy records are not returned to the program; therefore, a check does not need to be made for them.

```

REPORT02 ; UPDATE BDAM FILE IN PLACE

FILE UACCTS FB 80 STATUS USTAT
CODE 1 ; U = UPDATE, A = ADD, D = DELETE
UACCTNO 7 NU (ACCOUNT NUMBER)
UNAME 25
UADD1 25
UADD2 25

FILE DACCTS BDAM F 352 IO RANDOM DAKEY NU
KEYLEN 7 RELTRK DATT STATUS DASTAT COUNT DACNT
DACCTREC 352
DELFLAG 1 1 ;X'FF' MEANS RECORD HAS BEEN LOGICALLY DELETED
ACCTNO 7 4 NU
NAME 25 85
ADD1 25 110
ADD2 25 135

WORKAREA
PRIME 1 BI VALUE 29 ; 30 TRACKS ALLOCATED TO FILE
QUOTIENT 4 BI
REMAINDER 1 BI

READ UACCTS
DOWHILE USTAT NE 'E' ; READ RECORDS UNTIL END OF FILE
QUOTIENT = UACCTNO / PRIME ; USE PRIME NUMBER DIVISION
REMAINDER = UACCTNO - (QUOTIENT * PRIME)
MOVE REMAINDER TO DATT ; REMAINDER TECHNIQUE TO
MOVE UACCTNO TO DAKEY ; CALCULATE TRACK NUMBER
READ DACCTS
CASE DASTAT

```

Figure 605 [Example 3](#) Part 1 Output of Random Update, Fixed-Length Record with a Key

After the backup of the BDAM file, you get into the actual update. This part of the program randomizes the key and issues a read against the BDAM file to determine if the record exists on the BDAM file.

```

WHEN EQ 'N' ;RECORD NOT ON BDAM FILE
CASE CODE
  WHEN EQ 'A' ; ADD RECORD TO BDAM FILE
    MOVE SPACES TO DACCTREC
    MOVE UACCTNO TO DAKEY
    MOVE UACCTNO TO ACCTNO
    MOVE UNAME TO NAME
    MOVE UADD1 TO ADD1
    MOVE UADD2 TO ADD2
    WRITE DACCTS
    IF DASTAT EQ 'N'
      DATT = DATT + 1 ; ALLOW OVERFLOW TO
      WRITE DACCTS ; NEXT TRACK ONLY
      IF DASTAT EQ 'N'
        LIST UACCTNO 'FILE IS FULL' DATT
        MOVE 400 TO DYLCOUNT
        EXITDO
      ENDIF
      DYLCOUNT2 = DYLCOUNT2 + 1
    ENDIF
    LIST UACCTNO 'ADDED'
    DYLCOUNT4 = DYLCOUNT4 + 1
  ELSE ; UPDATE OR DELETE BDAM RECORD
    LIST UACCTNO 'MISSING'
    DYLCOUNT5 = DYLCOUNT5 + 1
  ENDCASE

```

Figure 606 [Example 3](#) Part 2 Output of Random Update, Fixed-Length Record with a Key

This part of the program handles the condition when the record is not on the BDAM file.

Addition of a record to the file is handled in the same way as when adding a record to the file in a random create.

```
WHEN EQ 'Y'                                ; RECORD FOUND ON BDAM FILE
  IF DELFLAG NE X'FF'                      ; CHECK LOGICALLY DELETED RECORD
CASE CODE
  WHEN EQ 'U'                              ; UPDATE RECORD ON BDAM FILE
    IF UNAME NE BLANKS
      MOVE UNAME TO NAME
    ENDIF
  IF UADD1 NE BLANKS
    MOVE UADD1 TO ADD1
  ENDIF
  IF UADD2 NE BLANKS
    MOVE UADD2 TO ADD2
  ENDIF
  WRITE DACCTS
  LIST UACCTNO 'UPDATED'
  DYLCOUNT6 = DYLCOUNT6 + 1
WHEN EQ 'D'                                ; DELETE RECORD ON BDAM FILE
  MOVE X'FF' TO DELFLAG                    ; LOGICALLY DELETE RECORD
  WRITE DACCTS
  LIST UACCTNO 'DELETED'
  DYLCOUNT3 = DYLCOUNT3 + 1
ELSE                                         ; ADD RECORD TO BDAM FILE
  LIST UACCTNO 'DUPLICATE'
  DYLCOUNT7 = DYLCOUNT7 + 1
ENDCASE
ELSE                                         ; LOGICALLY DELETED RECORD
CASE CODE
  WHEN EQ 'A'                              ; ADD RECORD TO BDAM FILE
    MOVE SPACES TO DACCTREC
    MOVE UACCTNO TO DACCTREC
    MOVE UNAME TO NAME
    MOVE UADD1 TO UADD1
    MOVE UADD2 TO UADD2
    WRITE DACCTS
    LIST UACCTNO 'ADDED'
    DYLCOUNT4 = DYLCOUNT4 + 1
  ELSE                                       ; UPDATE OR DELETE BDAM RECORD
    LIST UACCTNO 'MISSING'
    DYLCOUNT5 = DYLCOUNT5 + 1
  ENDCASE
ENDIF
```

Figure 607 [Example 3](#) Part 3 Output of Random Update, Fixed-Length Record with a Key

Part 3 of the program handles the condition when the record is on the BDAM file, whether or not it is logically deleted.

```
ELSE                                         ; INVALID REQUEST
  LIST UACCTNO 'RANDOMIZING ERROR' DAT1
  MOVE 401 TO DYLRURN
  EXITDO
ENDCASE
READ UACCTS
ENDDO
MOVE 'DACCTS' TO DYLCNAME1
MOVE DACNT TO DYLCOUNT1
MOVE 'OVERFLOW' TO DYLCNAME2
MOVE 'DELETED' TO DYLCNAME3
MOVE 'ADDED' TO DYLCNAME4
MOVE 'MISSING' TO DYLCNAME5
MOVE 'UPDATED' TO DYLCNAME6
MOVE 'DUPLICATE' TO DYLCNAME7
STOP
```

Figure 608 [Example 3](#) Part 4 Output of Random Update, Fixed-Length Record with a Key

Part 4 of the program handles an error condition and termination of the program.

The DYLCOUNT counters keep track of the update counts. They are shown in [Figure 609](#).

```

VISION:RESULTS CONTROL TOTALS

FILE      RECORD      CHARACTER      BLOCK      DROPPED
ID        COUNT        COUNT        COUNT      BLOCK COUNT

UACCTS          6          480

                        RECORDS                        PAGES

FILE PRINT                                1
REPORT PRINT                             1
FIXED BLANK COUNT
FIXED DECIMAL DIVIDE
DACCTS                                3
OVERFLOW
DELETED                                1
ADDED                                  1
MISSING                               2
UPDATED                               1
DUPLICATE                              1

```

Figure 609 [Example 3](#) Totals of Output of Random Update, Fixed-Length Record with a Key

BDAM file count of 3 is derived from deleted (actually an update), added, and updated counts.

```

ACCOUNT NUMBER

2007177      UPDATED
7777777      MISSING
8888888      MISSING
6123031      DELETED
6666666      ADDED
7012837      DUPLICATE

```

Figure 610 [Example 3](#) Report of Output of Random Update, Fixed-Length Record with a Key

This report shows the update activity.

The contents of the records in the VACCTS file are as follows:

```

      1      2      3      4      5      6      7
12345678901234567890123456789012345678901234567890123456789012345

U2007177HART,GARY
U7777777JONES,FRED
D8888888
D6123031
A6666666BAKER,JOHN      1234 MAIN STREET      ANYTOWN, CA 91023
A7012837JONES,FRED

```

Figure 611 [Example 3](#) Contents of Output of Random Update, Fixed-Length Record with a Key

Example 4 Reorganize File, Fixed-Length Record with a Key

```

OPTION STRUCTURED

REPORT01      ; COPY BDAM FILE RECORDS TO A TEMPORARY FILE

FILE DACCTS BDAM F 352 SEQUENTIAL
  KEYLEN 7 RELTRK DATT STATUS DASTAT COUNT DACNT
  DACCTREC 352
  DELFLAG 1 1 ; X'FF' MEANS RECORD HAS BEEN LOGICALLY DELETED

FILE TACCTS FB 352 OUTPUT FROM TACCTS
  TACCTREC 352

READ DACCTS
DOWHILE DASTAT NE 'E'      ; READ RECORDS UNTIL END OF FILE
  IF DASTAT NE 'D' AND    ; BYPASS DUMMY RECORDS AND
    DELFLAG NE X'FF'      ; LOGICALLY DELETED RECORDS
    MOVE DACCTREC TO TACCTREC
  WRITE TACCTS
ENDIF
READ DACCTS
ENDDO
MOVE 'DACCTS' TO DYLCNAME1
MOVE DACNT TO DYLCOUNT1
STOP

```

Figure 612 [Example 4](#) Reorganize File, Fixed-Length Record with a Key

The first part of the program copies the valid records to a temporary file and counts them.

In VSE, dummy records are not returned to the program; therefore, a check need not be made for them.

Control Totals of Example 4

VISION:RESULTS CONTROL TOTALS				
FILE ID	RECORD COUNT	CHARACTER COUNT	BLOCK COUNT	DROPPED BLOCK COUNT
TACCTS	200	70,400		
		RECORDS		PAGES
FILE PRINT				1
REPORT PRINT				1
FIXED BLANK COUNT				
FIXED DECIMAL DIVIDE				
DACCTS				201

Figure 613 [Example 4](#) Totals Part 1 of Reorganize File, Fixed-Length Record with a Key


```

REPORT02                ; BUILD NEW BDAM FILE ON TOP OF OLD FILE
                        ; USING RECORDS COPIED TO TEMPORARY FILE
FILE DACCTS BDAM F 352 OUTPUT FROM DACCTS RANDOM DAKEY NU
KEYLEN 7 RELTRK DATT STATUS DASTAT COUNT DACNT
DACCTREC 352

FILE TACCTS FB 352 STATUS TSTAT
TACCTREC 352
TACCTNO 7 4 NU

WORKAREA
PRIME 1 BI VALUE 29 ; 30 TRACKS ALLOCATED TO FILE
QUOTIENT 4 BI
REMAINDER 1 BI

READ TACCTS
DOWHILE TSTAT NE 'E' ; READ RECORDS UNTIL END OF FILE
QUOTIENT = TACCTNO / PRIME ; USE PRIME NUMBER DIVISION
REMAINDER = TACCTNO - (QUOTIENT * PRIME) ; REMAINDER TECHNIQUE TO
MOVE REMAINDER TO DATT ; CALCULATE TRACK NUMBER
MOVE TACCTNO TO DAKEY
MOVE TACCTREC TO DACCTREC
WRITE DACCTS
IF DASTAT EQ 'N'
DATT = DATT + 1 ; ALLOW OVERFLOW TO NEXT
WRITE DACCTS ; TRACK ONLY
IF DASTAT EQ 'N'
LIST TACCTNO 'FILE IS FULL' DATT
MOVE 400 TO DYLRRETURN
EXITDO
ENDIF
DYLCOUNT2 = DYLCOUNT2 + 1
ENDIF
IF DASTAT EQ 'I'
LIST TACCTNO 'RANDOMIZING ERROR' DATT
MOVE 401 TO DYLRRETURN
EXITDO
ENDIF
READ TACCTS
ENDDO
MOVE 'DACCTS' TO DYLCNAME1
MOVE DACNT TO DYLCOUNT1
MOVE 'OVERFLOW' TO DYLCNAME2
STOP

```

Figure 614 [Example 4](#) Totals Part 2 of Reorganize File, Fixed-Length Record with a Key

The second part of the program recreates the BDAM file on top of the old file using the valid BDAM records from the temporary file.

Control Totals of Example 4

VISION:RESULTS CONTROL TOTALS				
FILE ID	RECORD COUNT	CHARACTER COUNT	BLOCK COUNT	DROPPED BLOCK COUNT
TACCTS	200	70,400		
		RECORDS		PAGES
FILE PRINT				1
REPORT PRINT				
FIXED BLANK COUNT				
FIXED DECIMAL DIVIDE				
DACCTS			200	

Figure 615 Example 4 Totals of Reorganize File, Fixed-Length Record with a Key

Chapter 23: Using PDS and SSL Support

VISION:Results supports retrieval from and update of partitioned data sets (PDS for z/OS) and retrieval only from source statement libraries (SSL for VSE). For a description of PDS support, see [Using the PDS Function on page 612](#). For a description of SSL support, see [Source Statement Library Support on page 633](#).

VISION:Results supports direct access of z/OS partitioned data sets. You can selectively extract, interrogate, rename, alias, and print PDS members. You can also selectively update, copy, and add new members, as well as read and change PDS directory entries.

Applications of the PDS function include:

- Listing load module attributes in a format you specify.
- Converting libraries.
- Listing members of PROCLIB, MACLIB, and so on, in member name sequence.
- Backing up PROCLIB, MACLIB to tape.
- Giving alias names to PDS members.
- Copying and updating existing members.
- Adding and replacing members.

The program examples at the end of this chapter illustrate many of these techniques.

When a PDS function is used, member records or directory records are returned, depending on the type of command issued. The PDS function can access both directory records and member records within the same program.

Using the PDS Function

The PDS function is defined using a FILE statement. A program can contain an unlimited number of PDS files, restricted only by the amount of usable memory. The PDS function can handle up to 50 libraries that are concatenated together using JCL, subject to the constraints described in the appropriate IBM documentation.

Specific directory records can be read starting with the first record in the directory. Similarly, member records are read sequentially from the first record in the member. Use the READMEM command to read a member record. Use the READDIR command to read a directory record.

To write to a member record, use WRITEMEM. Use WRITEDIR to write to a directory record. All information contained in the member record and all information contained in the directory record are available to the program for processing purposes.

Input to VISION:Results

The PDS FILE statement has the same format as the VISION:Results FILE statement, with the addition of the keywords PDS, MEMBER, and NEWNAME. The complete format of the PDS FILE statement is:

```
FILE filename
PDS dataname MEMBER dataname [NEWNAME dataname] STATUS dataname
[INPUT | OUTPUT FROM filename]
[{FB | VB | F | V | U} [recordsize [blocksize]]]
[LENGTH dataname]
[COUNT dataname]
```

Figure 616 Input to VISION:Results

FILE

This entry is required. The command FILE identifies this statement as a file definition. The file name is required. It is used in the READDIR, READMEM, WRITEDIR, and WRITEMEM statements to specify the file the records are read from or written to. The file name is also the ddname used in the JCL. For example, if you have the FILE statement:

```
FILE SRCLIB PDS GENINFO FB 80 MEMBER MEMNAME
STATUS STCOND
```

then the DD statement for this file is:

```
//SRCLIB DD DSN=My.Source.Library,DISP=SHR
```

PDS Dataname

The PDS keyword is required to define the file as a PDS organized file. The required data name is the name of the data set ID area, a 50-byte field that contains the following information after the file is opened:

Contents	Size	Location
DSNAME	44	1
VOLID	6	45

To access the fields within the data set ID area, move the data set ID area to a field in a work area that has the subordinate fields defined. For an example, see [List All Member Names of a PDS \(Columns in Horizontal Alphabetic Sequence\) on page 621](#).

MEMBER Dataname

This entry is required. The data name is the name of an 8-byte field that contains the name of the member to be processed. When the READDIR command is executed, the member name associated with the directory record read is automatically placed in the MEMBER dataname. Blanks must be moved to the MEMBER field before each execution of the READDIR command.

To read or update a specific member, move the name of that member into the MEMBER dataname and, upon the next execution of the READMEM or WRITEMEM command, a record is read from, or a change is written to, that member.

STATUS Dataname

Required. The data name is the name of a 1-byte field that contains the condition code returned from a READMEM, READDIR, WRITEMEM, or WRITEDIR command. Check the STATUS dataname after each read or write command is executed. The condition codes are as follows.

Code	Description
Y	The command was successfully executed.
E	After a READMEM command, the E status means no more records exist in the member being processed. After a READDIR command, the E status means no more directory records exist, that is, end-of-directory has been reached. Setting the STATUS dataname to a blank after an E is set allows continued file access. Blanking the STATUS dataname after a CLOSE statement also allows further file access.
N	After a READMEM command, the N status means that the member you are attempting to read could not be found in the PDS library. After a WRITEMEM command, the N status means that the write could not be done.

FB|VB|F|V|U recordsize blocksize

The record format and record size are required; the block size is optional. Record format, recordsize, and blocksize are compared to the corresponding data found in the DCB at the time the file is opened. If different, processing is terminated. Recordsize is required to resolve the compiler's allocation of record space in memory. For undefined records (load libraries) with 0 record length, blocksize can be used instead of record length.

LENGTH Dataname

Optional for fixed-length records. Required for variable-length or undefined-format records. The data name is a 2-byte binary field that contains the length of the record.

COUNT Dataname

Optional. The data name is a 5-byte packed field that is incremented by 1 each time a READMEM or WRITEMEM command for the given file is executed. Within your program, you can re-initialize the COUNT dataname whenever convenient (for example, at the beginning of each member).

NEWNAME Dataname

Optional. The data name is the name of an 8-byte field to which an alias name for an existing PDS member or a new name for an existing PDS member is moved for processing. It is used in conjunction with either the ALIAS or CHANGE keywords and is only specified if the file has been opened as output. If coded, NEWNAME must immediately follow the MEMBER dataname in the PDS file definition.

INPUT

Default. The PDS file is opened as input.

OUTPUT FROM Filename

PDS files can only be output from themselves. The record must be written from its own output hold area after the record is moved to this area during the program logic.

READMEM and READDIR Commands

The READMEM and READDIR commands function in the same way as the READ command. The READMEM command reads member records sequentially until end-of-member occurs, passing back one record per READMEM execution. The READDIR command reads directory records sequentially until end-of-directory occurs, passing back one directory record per READDIR execution. After reading all the records from a member, the member name is blanked out automatically. The READMEM command is coded as follows:

```
READMEM filename
```

where filename is the name (as defined in the FILE statement) of the PDS file containing the members to be read. The member record returns to the file area implicitly or explicitly defined directly under the FILE statement. Examples in this chapter demonstrate this concept.

The READDIR command is coded as follows:

```
READDIR filename [dirarea]
```

where filename is the name (as defined in the FILE statement) of the PDS file from which directory records are to be read. The directory area (dirarea) is optional. If omitted, the READDIR command places the member name in the directory record into the MEMBER dataname and the actual directory record is not made accessible to the VISION:Results program.

When the directory area (dirarea) is present in the READDIR command, the member name in the directory area is moved into the MEMBER dataname and the directory record is moved into the directory area. If you only read the directory area sequentially, after completion, the member name should be blanked out manually.

If you process both the directory entry and member records, it is not necessary to blank the member name out. The directory area must be defined in a VISION:Results work area as a 75-byte field. The format of the PDS directory record is shown in [PDS Directory Format on page 632](#).

WRITEMEM and WRITEDIR Commands

The WRITEMEM and WRITEDIR commands function in the same way as the WRITE command. The WRITEMEM command writes information to a specified PDS member. The WRITEDIR command writes information to a specified PDS directory. The WRITEMEM command is coded as follows:

```
WRITEMEM filename
```

where filename is the name (as defined in the FILE statement) of the PDS file to which information is to be written.

The WRITEDIR command is coded as follows:

```
WRITEDIR filename [dirarea]  
[ADD|ALIAS|CHANGE|REPLACE|SCRATCH]
```

where filename is the name (as defined in the FILE statement) of the PDS file to which information is to be written. As in the READDIR command, the directory area (dirarea) is optional. If specified, information can be written to the PDS directory user area.

The STATUS data name can be checked after you have issued the WRITEDIR command. If the status code is Y, the WRITEDIR was successful; otherwise, the status code is set to N.

PDS Output Rules and Recommendations

- A maximum of 16 output files can be open at any one time in your program.
- The same data set cannot be referenced in more than one PDS FILE OUTPUT statement per program.
- Because you are responsible for file recovery when a PDS file is opened as output, open all PDS output files as DISP=OLD. File integrity can be compromised if PDS output files are opened as DISP=SHR.

ADD Keyword

The ADD keyword writes a new entry to the PDS directory. To use the command:

- Collect member records.
- Move the new entry name to the MEMBER field.
- Issue the WRITEMEM file name command to write the new member. At least one WRITEMEM command must be issued before the WRITEDIR command can be issued.
- Issue the WRITEDIR filename ADD command to write a new directory entry.

The ADD keyword is coded as follows:

```
WRITEDIR filename ADD
```

where filename is the name of the PDS file to which you are adding a new entry.

If you attempt to ADD an entry that is already in the directory, the command is ignored and an N is returned in the STATUS field.

ALIAS Keyword

The ALIAS keyword writes an additional entry to the PDS directory and gives a new name (an alias) to a PDS member. The member can then be referenced using either the alias or the member name.

To use the ALIAS keyword command

1. Code the NEWNAME keyword and an associated user-supplied data name in the PDS file statement.
2. Move the alias name to that field.
3. Move the name of the member you want to alias to the MEMBER field.

Both moves must be done before issuing the WRITEDIR command.

The ALIAS keyword is coded as follows:

```
WRITEDIR filename ALIAS
```

where filename is the name of the PDS file containing the member you want to alias.

CHANGE Keyword

The CHANGE keyword writes an entry to the PDS directory and changes the name of a PDS member. After the CHANGE has been made, the member can only be accessed using the new name.

To use the **CHANGE** keyword command

1. Code the NEWNAME keyword and an associated user-supplied data name in the PDS file statement.
2. Move the new member name to that field.
3. Move the old member name to the MEMBER field.

Both moves must be done before issuing the WRITEDIR command.

The CHANGE keyword is coded as follows:

```
WRITEDIR filename CHANGE
```

where filename is the name of the PDS file containing the member you want to rename.

REPLACE Keyword

The REPLACE keyword writes a directory entry. Use it when you have updated the contents of a PDS member and want to reference the updated version with the same name. After you issue the REPLACE command, only the updated version of the member can be accessed.

To use the **REPLACE** command

1. Move the entry name to the MEMBER field.
2. Make any needed changes to the member (this may require issuing the READMEM command to read the PDS member).
3. Issue the WRITEMEM file name command to write the new member.
4. Issue the WRITEDIR file name REPLACE command. The directory entry now points to the new member.

The REPLACE keyword is coded as follows:

```
WRITEDIR filename REPLACE
```

where filename is the name of the PDS file containing the entry name you want to reassign.

If you attempt to execute a REPLACE where the entry name is not in the directory, the entry is added to the directory but the STATUS field is set to N.

If OPTION PDSREPA or the DYLINSTL parameter PDSREPL=Y has been specified, the STATUS field is set to A.

SCRATCH Keyword

The SCRATCH keyword deletes an entry from the PDS directory. After the SCRATCH is executed, the PDS member is no longer accessible. To use the command, move the entry name you want to delete to the MEMBER field before issuing the WRITEDIR command.

The SCRATCH keyword is coded as follows:

```
WRITEDIR filename SCRATCH
```

where filename is the name of the PDS file containing the entry name you want to delete.

To terminate reading the directory at any time and to re-initiate it at the beginning of the directory file

1. Issue a CLOSE filename command that closes the file.
2. Move a blank into the STATUS data name.

When the next READDIR command is executed, the first record in the directory file is read.

WRITEDIR and WRITEMEM commands can only be used with files opened for output. READDIR and READMEM commands are limited to input files.

PDS Examples

The following examples illustrate some potential uses of the PDS function. After the narrative portion for each example, the SYSOUT of the described example is provided. For a complete description of the statements used, see the specific chapter in this manual.

Example 1 Print Three Members

In this example, the members OPEN, CLOSE, and GETMAIN are printed in their entirety from SYS1.MACLIB. No directory information is included in the report.

The numbers at the left of the following example are for reference purposes only.

The PDS file is defined in Statement 1. The data set ID area data name must always be defined after the keyword PDS, even though you do not use it elsewhere in your program. The name of the member to read is moved into the data name MEMNAME at the appropriate times. In Statement 2, MEMREC is the name of the field into which the member records are read.

Statements 3 through 8 control the reading and printing of each member by moving the member name into MEMNAME and performing the read-print logic. The execution of the program is terminated in Statement 9. The read-print logic is in Statements 10 through 21.

The reading of each member record is invoked in Statement 11. If the status byte returned is a Y denoting successful execution, the member record is listed and the logic branches back to read another member record. If the status byte returned is an N, the requested member did not exist in the library and the member name is printed with the message NOTFOUND. If the status returned was neither blank nor N, the program assumes that it was E, for end-of-member, and exits the performed logic.

```

1  FILE MEMFILE FB 80 PDS  DSNAREA  MEMBER MEMNAME
   STATUS EOFIN
2  MEMREC      80

3  MOVE 'CLOSE' TO MEMNAME
4  PERFORM READWRIT TO READWRITX
5  MOVE 'GETMAIN' TO MEMNAME
6  PERFORM READWRIT TO READWRITX
7  MOVE 'OPEN' TO MEMNAME
8  PERFORM READWRIT TO READWRITX
9  STOP

10 READWRIT:
11  READMEM MEMFILE
12  IF EOFIN EQ 'Y'
13      LIST SUPPRESS MEMNAME (MEMBER NAME)
        MEMREC (CONTENTS OF MEMBER)
14  GOTO READWRIT
15  ENDIF
16  IF EOFIN EQ 'N'
17      MOVE 'NOTFOUND' TO DYLPRTCOMM
18      PRINT MEMNAME
19  ENDIF
20  MOVE SPACE TO EOFIN
21  READWRITX:

22  T1 'COMPUTER ASSOCIATESTEST '
   'DYL PDS-EXAMPLE 1'
   WITH 2 AFTER

```

Figure 617 [Example 1](#) Print Three Members

```

                                COMPUTER ASSOCIATES TEST DYL PDS-EXAMPLE 1
                                CONTENTS OF MEMBER
MEMBER NAME                      CLOSE
CLOSE                           MACRO                                00020000
                                &CNVME  CLOSE &PAR1,&MF-I,&TYPE=,&MODE=    @02C 00021510
                                *                                           00023010
                                *CLOSE MACRO                                @L1A 00026010
                                *                                           @L1A 00029010
                                *$MAC(CLOSE):                                @L1A 00032010
                                *                                           @L1A 00035010
                                *   COPYRIGHT  TYPE=ASMMAC,                @L1A 00036210
                                *   CLASS=EST,                                @L1A 00037410
                                *   LEVEL=(HDP2230),                        @L1A 00038610
                                *   SECSMT='IBM INTERNAL USE ONLY'          @L1A 00039810
                                *                                           @L1A 00041010
                                * STATUS = MVS/XA DFP RELEASE 2.3            @L1C 00042010
                                *                                           @L1A 00043010
                                *CHANGE ACTIVITY - AS FOLLOWS:              @L1A 00044010
                                *                                           * 00047010
                                *$L1=ACB31BIT,HDP2230,,STLSS: O/C/E SUPPORT OF 31-BIT VSAM @L1A 00050010
                                *$01=OY05946,HDP2230,,NSDRJV: EXTRA INSTRUCTION WITH MODE=31 @01A 00051010
                                *$02=OY07793,HDP2230,870720,STLSS: REMOVE SELEVEL DEPENDENCE @02A 00052010
                                *                                           * 00053010
                                * ***** **                                * 00054010
                                *                                           @02D 00056010
                                *                                           00060000
                                *   LCTA  &ACTR,&ASUM,&CTR,&CTR1,&LGTH,&NUMBER    @L1C 00080010
                                *   LCTE  &MFESW,&MFLSW,&MFI2SW,&NULLSW,&B(5),&MSW 00100000
                                *   LCLC  &PARA                                @02D 00110010
                                *                                           00120000
                                *NUMBER SETA  N' &PAR1                        @02D 00130010
                                *                                           00140000
                                *CTR    SETA  1

```

Figure 618 [Example 1](#) Output—Print Three Members (Page 1 of 2)

```

&CTRL SETA 2 00160000
AIF (N'&SYSLIST LE 1).TYPE IF ONLY 1 (PARMLIST) A45585 00185000
IHERMAC 238 EXCESSIVE POS. PARMS A45585 00190000
.TYPE AIF (T'&TYPE NE 'O' AND ' &TYPE' NE 'T').ERROR8 A45585 00195000
AIF (T'&MODE EQ '0').MFTST @LIA 00199110
AIF ('&MODE' NE '31' AND '&MODE' NE '24').ERROR9 @LIA 00203210
AIF ('&MODE' EQ '24').MFTST @LIA 00207310
&MSW SETB 1 SET MODE=31 SWITCH @LIA 00211410
.MFTST AIF ('&MF' EQ 'L').RIEL @LIC 00215510
AIF ('&MF' EQ 'I').TESTI 00220000
AIF (N'&MF LE 1).ERROR3 00240000
AIF ('&MF(1)' NE 'E').ERROR3 00260000
&CNAME IHEINNA &MF(2) 00280000
* 00293310
AIF ('&PAR1' EQ '').RGFLIP @LIC 00306610
AIF ('&PAR1 (&NUMBER)' NE '').LOOPB 00320000
&NUMBER SETA &NUMBER-1 00340000
AIF (&NUMBER EQ 0).RGFLIP 00360010
AGO .LOOPA 00380000
AIF ('&PAR1 (&CTR)' NE '').TESTCTR 00400000
&NULLSW SETB 1 00420000
.TESTCTR AIF (&CTR EQ &NUMBER).QUITNOW 00440000
AIF ('&PAR1 (&CTRL)' EQ '').NULLTWO 00460000
&MFESW SETB 1 00480000
AGO .CLRSW 00500000
.ERITRIN AIF (&CTRL EQ &NUMBER).THRUNOW 00580000
AIF (&NULLSW).NOFIRST 00600000
AIF ('&PAR1 (&CTR)' (1,1) EQ ' ').REGISTER 00620000
LA 14,&PAR1 (&CTR) PICK UP DCB ADDRESS 00640000
AIF (NOT &MSW).MODE24A @LIA 00646610
&ACTR SETA &ACTR+4 @LIA 00653210
.MODE24A ST 14,&ACTR. (1,0) STORE INIO LIST @LIC 00659810

```

Figure 618 Example 1 Output—Print Three Members (Page 2 of 2)

Example 2 List All Member Names of a PDS (Columns in Horizontal Alphabetic Sequence)

This example lists the names of the members in the partitioned data set Z01.DY.DYLMAC. The names are printed 10 across in the report. In this example, the PDS data name area gets the volume serial number of the disk pack containing the Z01.DY.DYLMAC library.

The numbers at the left of the following example are for reference purposes only.

The member names are taken from the directory records. The PDS file is defined in Statement 1. The volume serial number is taken from the data set ID area field DSNAREA. The READDIR command places the member names into the field MEMNAME. The name of the status indicator is STATIND. Because you do not have access to the individual fields in the data set ID area, you need to define in a work area a field WDSNAREA that maps the fields in the data set ID area (see Statements 2 through 6). The fields within the print line are defined in Statements 8 through 11.

The directory record is read using the READDIR command shown in Statement 12. The directory record area is not included in the READDIR command because you are only interested in the member name, which is available in MEMNAME. When the status returned becomes E, the directory file is at end; at this point, the last print line is written and the program is terminated. The information in the data set ID area is available after the first execution of the READDIR command. Therefore, the DSNAREA must be moved to the work area field once after the execution of the first READDIR command, as shown in Statements 15 through 18.

The remainder of the program is standard VISION:Results code that builds and prints the print line. The volume serial number appears in the second title line as the contents of the data set ID area field, WVOLID.

```

1  FILE LIBFILE FB 80 PDS DSNAREA MEMBER MEMNAME
   STATUS STATIND
2  WORKAREA
3  WDSNAREA 52 1
4  WDSNAME 44 1
5  WVOLID 6
6  WRECLNG 2 BI

7  WORKAREA
8  MEMBNAMES 100
9  M1 8 1 M2 8 9 M3 8 17 M4 8 25
10 M5 8 33 M6 8 41 M7 8 49 M8 8 57
11 M9 8 65 M10 8 73

12 READDIR LIBFILE
13 IF STATIND EQ 'E' PERFORM LISTDET TO LISTEND
14 STOP ENDIF
15 ON ONE
16 MOVE DSNAREA TO WDSNAREA
17 MOVE 0 TO INW
18 ENDONE
19 MOVE MEMNAME TO M1 (INW)
20 IF INW LT 72 INW = INW + 8 ACCEPT ENDIF
21 LISTDET:
22 LIST M1 (MEMBER'NAME) M2 (MEMBER'NAME)
      M3 (MEMBER'NAME) M4 (MEMBER'NAME)
      M5 (MEMBER'NAME) M6 (MEMBER'NAME)
      M7 (MEMBER'NAME) M8 (MEMBER'NAME)
      M9 (MEMBER'NAME) M10 (MEMBER'NAME)
23 MOVE 0 TO INW
24 MOVE SPACES TO MEMBNAMES
25 LISTEND:

26 T1 'COMPUTER ASSOCIATES TEST'
   'DYL PDS-EXAMPLE 2'
27 T2 'VISION:RESULTS SUPPORT UTILITY---XXXXXX'
28 T2+77 WVOLID
29 T3 'COMPUTER ASSOCIATES MACLIB' WITH 2 AFTER

```

Figure 619 [Example 2](#) List All Member Names of a PDS (Columns in Horizontal Alphabetic Sequence)

```

COMPUTER ASSOCIATES TEST DYL PDS-EXAMPLE 2
VISION:RESULTS SUPPORT UTILITY---TSO005
COMPUTER ASSOCIATES MACLIB

MEMBER MEMBER MEMBER MEMBER MEMBER MEMBER MEMBER MEMBER MEMBER
NAME NAME NAME NAME NAME NAME NAME NAME NAME NAME
AADR000 AADR000 AADR000 AADR000 AIPSW00 ALCNF00 ALDRV00 ALFQA00 APERR00 APSMP00
AREED00 ARCEB00 ARCP00 ARCP00D ARCP10D ARDELDAD ARDRV05 AREND00D ARERR00 ARFF00
ARIC00 ARIC00D ARPHTAB ARPHOOD ARPUB00D ARSPY00 ARSPY00D ARSROOD ASEND00 ASMNONE
ASPSNCOM ASPSN00 ASRNG00 ASRPS00 AVAGE00 AVCNF00 AVCNW00 AVDRV00 AVFJG00 AVFRQ00
AVRPS00 AVRNR00 AVRSR00 AVSBI00 AVSRN00 AVSRT00 AVSSM00 AVIMG00 AWCNR00 AWCIL00
AVDRV00 AWDYLOO AWIOO0D AZDRV00 AZSPY00 DAENRAT DAENRAT DAENCOM DAENDAT DAENDIV
DAENDFQ DAENDHS DAENDMY DAENDPS DAENDGP DAENDST DAENDVR DAENDWS DAENPKF DAENPSN
DAENSRT DAENSRT2 DAENDVR DAINITZ DAINLOAD DAINRFLD DAINSCB DAINSCN DAPSAG DAPSON
DAPSEQ DAPSRAN DAPSSP DARTDEW DARTDID DARTDVR DARTFPT DARTIMD DARTIHT DARTPRID
DARTSED DARTSPY DARTSPYD DYLATAB DYLAUDA DYLAUDB DYLAUDC DYLDECRY DYLDUMMY DYLENCRY
DYLMTUG DYLPSEVC DYLPSEVS DYLPSEVJ DYLRAND DYLRAND2 DYLSMFTC DYLSNDCD DYLSORT LJJFFZZZZ
LJFVZZZZ LJFVZZZZ LJGF0ZZZ LJGVZZZZ LJGVZZZZ LJVP0BID IVCNCE IVFQCB NEGTRY TRYTIME
XXXX

```

Figure 620 [Example 2](#) Output—List All Member Names of a PDS (Columns in Horizontal Alphabetic Sequence)

Example 3 Backup to Tape Members of a PDS and Add IEBUPDTE Control Statements in Front of Each Member

In this example, all members with names starting with the prefix LOW are read from the macro library SYS1.MACLIB and written to tape. An IEBUPDTE ADD control statement is written in front of each member. This is done so that these members can be restored using the IBM IEBUPDTE utility.

The IEBUPDTE control statement prints as each member is written to tape. This provides a list of the members backed up to tape.

The numbers at the left of the following example are for reference purposes only.

In this example, you are reading both directory records and member records in the program. The PDS file has been defined in Statement 1 and the member record field in Statement 2. The backup file is defined in Statement 3 and the backup record field in Statement 4.

Because you need access to just the first three characters of the member name in MEMNAME, provide a work area field into which you can move MEMNAME. That field is WMEMNAME and the prefix field is WMEMPRFX, as defined in Statements 6 and 7. The IEBUPDTE Statement is defined in Statements 9 through 11. At the appropriate time, the member name is moved from MEMNAME to the field SELECTED, which is a part of the IEBUPDTE statement.

The READDIR command in Statement 12 reads the directory record. You are only interested in the member name, so you have not included a directory area in the READDIR command. When the status returned becomes E, the program terminates. In Statements 14 and 15, the member name is tested for the prefix LOW; if the prefix is not LOW, the directory record is rejected and the flow of logic returns to the top of the program.

The IEBUPDTE statement is built, written, and printed in Statements 16 through 19. The READMEM command reads the member in Statement 21. A status of E means that end-of-member is reached and the program goes back to reading directory records. In Statements 23 and 24, the member records are written to tape.

```

1  FILE LIBFILE FB 80 PDS DSNFIELD MEMBER MEMNAME
2  STATUS STATIND
   MEMREC 80

3  FILE BACKFIL FB 80 OUTPUT FROM BACKFIL
4  BACKREC 80

5  WORKAREA
6  WMEMNAME  8  1
7  WMEMPRFX  3  1

8  WORKAREA
9  ADDREC    20  1
10 ADDNAME   12  1  VALUE './ ADD NAME='
11 SELECTED   8   VALUE ' '

12 READDIR LIBFILE
13 IF STATIND EQ 'E' STOP ENDIF
14 MOVE MEMNAME TO WMEMNAME

```

Figure 621 Example 3 Backup to Tape Members of a PDS and Add IEBUPDTE Control Statements in Front of Each Member (Page 1 of 2)

```
15     IF WMEMPRFX NE 'LOW' REJECT ENDIF
16     MOVE MEMNAME TO SELECTED
17     MOVE ADDREC TO BACKREC
18     WRITE BACKFIL
19     PRINT ADDREC

20 MEMBREAD:
21     READMEM LIBFILE
22     IF STATIND EQ 'E' REJECT ENDIF
23     MOVE MEMREC TO BACKREC
24     WRITE BACKFIL
25     GOTO MEMBREAD
```

Figure 621 Example 3 Backup to Tape Members of a PDS and Add IEBUPDTE Control Statements in Front of Each Member (Page 2 of 2)

```
./ ADD NAME=LOWUP
./ ADD NAME=LOWUP2
./ ADD NAME=LOWUP3
./ ADD NAME=LOWUP4
./ ADD NAME=LOWUP5
```

Figure 622 [Example 3](#) Output—Backup to Tape Members of a PDS and Add IEBUPDTE Control Statements in Front of Each Member

Example 4 List Selected Member Records of a PDS

This example lists selected member records from each member of the PDS file DYL.PUB.PROCLIB. You may need to do a global scan of a PDS file to locate multiple occurrences of a particular word or statement (for example, the members of a production PROCLIB that have STEPLIB DD statements included in the JCL stream).

The numbers at the left of the following example are for reference purposes only.

The PDS file is defined in Statement 2. The procedural logic consists of a DOWHILE loop for reading the directory entries. This loop starts at Statement 14 and ends at Statement 37. The directory entries are read using the READDIR commands at Statements 13 and 36. Note that the optional directory area is omitted in the READDIR commands because you do not need to access the actual directory records. The read directory loop terminates when the PDS status code is E, signifying that end of file was reached for reading directory entries.

For each directory entry read using the READDIR command, the member name is moved directly to the field defined by the data name MEM (Statement 2) and a DOWHILE loop reads each member record. This loop starts at Statement 22 and ends at Statement 34. The member records are read using the READMEM commands at Statements 21, 28, and 32. The read member loop terminates when the PDS status code is E, signifying that end of file was reached for reading member entries.

When a member record is read, the program looks for the literal //STEPLIB (Statement 23). If found, the statement and all of its concatenated data sets are listed. This is accomplished by the DOUNTIL loop at Statements 24 through 29.

After all the member records have been read, the status code is reset to blank (Statement 35) to prepare to continuing reading directory entries.


```

1  OPTION STRUCTURED
2  FILE LIBFILE  FB 80  PDS DSNAREA  MEMBER MEM  STATUS PDSSTAT
3  PDSREC      80  1
4  POS1       10  1
5  COMMAND     72  1
6  WORKAREA
7  PROCLINE    131  1  VALUE ' '
8  PROCNAME     8  2
9  PROCSTMT    80  15
10 WORKAREA
11 BLKLINE      1  VALUE 'N'
12 REPORT 48 LONG
13 READDIR LIBFILE
14 DOWHILE PDSSTAT NE 'E' ; READ DIRECTORY LOOP.
15 IF BLKLINE EQ 'Y'
16 MOVE SPACES TO PROCLINE
17 LIST PROCLINE ( )
18 ENDIF
19 MOVE 'N' TO BLKLINE
20 MOVE MEM TO PROCNAME
21 READMEM LIBFILE
22 DOWHILE PDSSTAT NE 'E' ; READ MEMBER LOOP.
23 IF POS1 EQ '//STEPLIB '
24 DOUNTIL POS1 NE '// ' OR PDSSTAT EQ 'E' ; PRINT LOOP.
25 MOVE PDSREC TO PROCSTMT
26 LIST PROCLINE ( )
27 MOVE SPACES TO PROCNAME
28 READMEM LIBFILE
29 ENDDO ; END PRINT LOOP.
30 MOVE 'Y' TO BLKLINE
31 ELSE
32 READMEM LIBFILE
33 ENDIF
34 ENDDO ; END MEMBER LOOP.
35 MOVE ' ' TO PDSSTAT
36 READDIR LIBFILE
37 ENDDO ; END DIRECTORY LOOP.
38 STOP
39 T1 'VISION:RESULTS PDS EXAMPLE 4' WITH 2 AFTER
40 T2 'COMPUTER ASSOCIATES PROCLIB' WITH 3 AFTER
41 T3 ' ' WITH 2 AFTER
42 T3+2 'PROCEDURE NAME'

```

Figure 623 [Example 4](#) List Selected Member Records of a PDS

```

VISION:RESULTS EXAMPLE 4
COMPUTER ASSOCIATES PROCLIB

PROCEDURE NAME
COBUCL      //STEPLIB DD DSN=SYS1.V4.COBLIB,DISP=SHR *VERSION 4 COMPILER & LIB*
COBUCLG     //STEPLIB DD DSN=SYS1.V4.COBLIB,DISP=SHR *VERSION 4 COMPILER & LIB*
CVDSKCMS    //STEPLIB DD DSN=CODLOAD,DISP=SHR
CVLODOBJ    //STEPLIB DD DSN=AUDLOAD,DISP=SHR
CVTAPCMS    //STEPLIB DD DSN=CODLOAD,DISP=SHR
CV280       //STEPLIB DD DSN=DUMYLOAD,DISP=SHR
              // DD DSN=BK28030,DISP=SHR
              // DD DSN=AUDLOAD,DISP=SHR

```

Figure 624 [Example 4](#) Output—List Selected Member Records of a PDS
(Page 1 of 2)

```
//STEPLIB DD DSN=DUMYLOAD,DISP=SHR
//          DD DSN=BK28030,DISP=SHR
//          DD DSN=BKAUD21F,DISP=SHR
//          DD DSN=AUDLOAD,DISP=SHR
//STEPLIB DD DSN=DUMYLOAD,DISP=SHR
//          DD DSN=BK28030,DISP=SHR
//          DD DSN=BKAUD21F,DISP=SHR
//          DD DSN=AUDLOAD,DISP=SHR
//STEPLIB DD DSN=DUMYLOAD,DISP=SHR
//          DD DSN=BK28030,DISP=SHR
//          DD DSN=BKAUD21F,DISP=SHR
//          DD DSN=AUDLOAD,DISP=SHR
//STEPLIB DD DSN=DUMYLOAD,DISP=SHR
//          DD DSN=BK28030,DISP=SHR
//          DD DSN=BKAUD21F,DISP=SHR
//          DD DSN=AUDLOAD,DISP=SHR
//STEPLIB DD DSN=DUMYLOAD,DISP=SHR
//          DD DSN=BK28030,DISP=SHR
//          DD DSN=BKAUD21F,DISP=SHR
//          DD DSN=AUDLOAD,DISP=SHR
//STEPLIB DD DSN=DUMYLOAD,DISP=SHR
//          DD DSN=BK28030,DISP=SHR
//STEPLIB DD DSN=MEN.MI.PUB.LOD,DISP=SHR
//STEPLIB DD DSN=MEN.MI.PUB.LOD,DISP=SHR
```

Figure 624 [Example 4](#) Output—List Selected Member Records of a PDS
(Page 2 of 2)

Example 5 Assign an Alias Name to a PDS Member

In this example, you assign a second directory name to PDS member TBL01 so that you can access it using either name.

The numbers at the left of the following example are for reference purposes only.

The PDS file is defined in Statement 1. You have coded NEWNAME and an associated user supplied dataname, NEWINFO, so that you can move the alias name to that field later in the program. You have also specified OUTPUT FROM, indicating that this PDS file is to be opened as output, and allowing us to write to the file.

Before you can issue the WRITEDIR command, move the entry name you want to alias to the MEMBER field and move the alias name to the NEWNAME field. These moves are accomplished in Statements 2 and 3.

Statement 4 writes the alias name to the directory. The member can now be accessed by specifying either TBL01 or RATETBL.

The procedure is the same if you want to use the CHANGE command to change the name of a PDS member. Move the old member name to the MEMBER field. Move the new member name to the NEWNAME field and issue the WRITEDIR file name CHANGE command.

```

1  FILE WRITPDS FB 80 PDS DSNAREA MEMBER OLDINFO
   NEWNAME NEWINFO OUTPUT FROM WRITPDS STATUS PDSSTAT

2  MOVE 'TBL01' TO OLDINFO
3  MOVE 'RATETBL' TO NEWINFO

4  WRITEDIR WRITPDS ALIAS

5  IF PDSSTAT EQ 'Y'
6    LIST 'TBL01 ALIAS: ' NEWINFO
7  ELSE
8    PRINT 'WRITE FAILED'
9  ENDIF
10 STOPRUN:
11  STOP
12  T1 'PDS EXAMPLE 5' WITH 3 AFTER

```

Figure 625 [Example 5](#) Assign an Alias Name to a PDS Member

```

PDS EXAMPLE 5

TBL01 ALIAS:          RATETBL

```

Figure 626 [Example 5](#) Output—Assign an Alias Name to a PDS Member

Example 6 Copy and Update a PDS File

In this program, you want to perform a global scan of a PDS file to locate all STEPLIB DD statements. As the file is read, each STEPLIB statement found is replaced with a new STEPLIB statement. The entire PDS file is copied, and all STEPLIB statements are updated.

The numbers at the left of the following example are for reference purposes only.

This program reads and writes both directory and member records. Two PDS file statements are coded that access the same PDS file. One is opened as input, the other as output (Statements 2 and 4).

Because you want to access the entire PDS directory record, you have defined it in a work area (Statements 8-13). DIRINFO is the data name that references the entire directory record. When you issue the READDIR command later in the program, the member name is moved to the MEMBER field and the directory record is moved into this directory area.

To accomplish the reading and writing of all directory and member records, two DOWHILE loops are used. The main loop processes all of the directory records, while the secondary loop is nested within the main loop to process all of the member records for each directory entry.

First, the directory is read (Statement 17). Then, the main DOWHILE loop is entered that processes all of the directory records (Statement 18). As each directory is read, the member name in the MEMBER field of the input file is moved to the MEMBER field of the output file for further processing (Statement 19). Because you are accessing the entire directory record, you can test the alias bit to determine whether or not the entry is an alias. If it is, a message is issued along with the member name and further processing of this directory entry is bypassed by exiting the main DOWHILE loop (Statements 20-23).

Next, the program issues the initial read of the member record and enters the secondary DOWHILE loop (Statements 24-25). This loop sequentially processes all member records for that particular directory entry. The input record is moved to the output record area and each record is tested for a match with the character string //STEPLIB. When a match is encountered, the STEPLIB statement is replaced with a new STEPLIB statement (Statements 15-16) and written to a report (Statements 26-31).

All member records, including the updated //STEPLIB statements, are written to the output file, OUTPDS (Statement 32). The status byte, WRITSTAT, is tested after each WRITEMEM command. If the status byte is not Y, the program is terminated. If it is Y, the next member record is read and the secondary loop continues until the entire member file is read and written. At this point, you exit the secondary loop and move a space to READSTAT to initialize it for the next directory read.

The WRITEDIR with REPLACE is then issued to write the directory entry only, which now points to the updated version of the member. If the WRITEDIR command fails, a message is printed and the program reads the next directory entry and continues this cycle until all members have been processed.

```

1  OPTION STRUCTURED

2  FILE INPDS FB 80 PDS DSNAREA MEMBER OLDMEM INPUT STATUS READSTAT
3      INREC          80 1

4  FILE OUTPDS FB 80 PDS GENINFO MEMBER NEWMEM STATUS WRITSTAT
5      OUTPUT FROM OUTPDS
6      OUTREC          80 1
7      FILESTMT        10 1

8  WORKAREA
9      DIRINFO          75 1
10     DIRNAME          8 2
11     DIRPNTR          3 BI
12     DIRALIAS          1 BI
13     DIRUSER          62

14 WORKAREA
15     NEWFILE          80
16     VALUE '//STEPLIB DD DSN=SYS3.PROD.LOADLIB,DISP=SHR'
17     READDIR INPDS DIRINFO
18     DOWHILE READSTAT NE 'E'
19         MOVE OLDMEM TO NEWMEM
20         IF DIRALIAS ONES X'80'
21             LIST OLDMEM 'MAY HAVE TO BE RE-ESTABLISHED'
22             EXITDO
23     ENDIF
24     READMEM INPDS
25     DOWHILE READSTAT NE 'E'
26         MOVE INREC LENGTH 80 TO OUTREC
27         IF FILESTMT EQ '//STEPLIB '
28             MOVE NEWFILE TO OUTREC LENGTH 80

```

Figure 627 Example 6 Copy and Update a PDS File (Page 1 of 2)

```

29          LIST NEWMEM (MEMBER NAME) 'BEFORE: ' INREC ( )
30          LIST ' ' AT NEWMEM 'AFTER: ' OUTREC ( ) WITH 2 AFTER
31          ENDIF
32          WRITEMEM OUTPDS
33          IF WRITSTAT NE 'Y'
34              PRINT 'MEMBER FAILED'
35              STOP
36          ENDIF
37          READMEM INPDS
38          ENDDO
39          MOVE SPACE TO READSTAT
40          WRITEDIR OUTPDS DIRINFO REPLACE
41          IF WRITSTAT NE 'Y'
42              PRINT 'DIRECTORY FAILED'
43          ENDIF
44          READDIR INPDS DIRINFO
45          ENDDO
46          STOP

47          T1 'VISION:RESULTS PDS EXAMPLE 6' WITH 3 AFTER

```

Figure 627 Example 6 Copy and Update a PDS File (Page 2 of 2)

```

VISION:RESULTS PDS EXAMPLE 6

MEMBER NAME

MEMBER1      BEFORE:    //STEPLIB DD DSN=SYS1.PROD.LOADLIB,DISP=SHR
              AFTER:    //STEPLIB DD DSN=SYS3.PROD.LOADLIB,DISP=SHR

MEMBER2      BEFORE:    //STEPLIB DD DSN=SYS1.PROD.LOADLIB,DISP=SHR
              AFTER:    //STEPLIB DD DSN=SYS3.PROD.LOADLIB,DISP=SHR

MEMBER3      BEFORE:    //STEPLIB DD DSN=SYS1.PROD.LOADLIB,DISP=SHR
              AFTER:    //STEPLIB DD DSN=SYS3.PROD.LOADLIB,DISP=SHR

MEMBER3      BEFORE:    //STEPLIB DD DSN=SYS1.PROD.LOADLIB,DISP=SHR
              AFTER:    //STEPLIB DD DSN=SYS3.PROD.LOADLIB,DISP=SHR

```

Figure 628 [Example 6](#) Output—Copy and Update a PDS File

Example 7 Add New Members to a PDS

In this example, records from a transaction file are read in and all records with the same distributor code are grouped and added to the PDS as new members.

The numbers at the left of the following example are for reference purposes only.

An initial read of the transaction file is issued and the main DOWHILE loop is entered (Statements 14-15). The distributor code, DISTCODE, is moved to the SAVECODE work area, and then the secondary DOUNTIL loop is entered (Statements 16-17). In this loop, as long as DISTCODE equals SAVECODE, the transaction records are moved to the output area and then written to the output PDS file. If the record cannot be written, a message is printed and the program stops (Statements 18-24).

The program then reads the next transaction record and checks to see if the distributor code has changed. This is checked at the end of the DOUNTIL loop. If the distributor code has changed, this means that a completely new member has been written, and the program exits the secondary loop.

The new member is added to the PDS directory by moving SAVECODE to NEWDATA (the member field of the PDS file) and issuing the WRITEDIR command with ADD (Statements 27-28). This adds the new member to the directory, giving it the name found in NEWDATA. If an N is returned in NEWSTAT following this command, it indicates that the entry that the program is attempting to add is already in the directory. If this occurs, the program lists the member name found in SAVECODE with a message and the program stops (Statements 29-32).

If the new member name has been successfully written to the directory, the program begins processing the next transaction record. This cycle continues until all transaction records have been read and written to the PDS file.

```
1  OPTION STRUCTURED
2  FILE TRANSIN FB 80 STATUS INSTAT
3      INREC      80  1
4      DISTCODE   3  1
5      DIST       25  5
6      FILM       35  36
7      GROSS      5  71 NU A
8      GUARANTY   5  76 NU A
9  FILE OUTPDS FB 80 PDS GENINFO MEMBER NEWDATA STATUS NEWSTAT
10     OUTPUT FROM OUTPDS
11     OUTREC      80
12  WORKAREA
13     SAVECODE    3
14  READ TRANSIN
15  DOWHILE INSTAT NE 'E'
16     MOVE DISTCODE TO SAVECODE
17  DOUNTIL DISTCODE NE SAVECODE
18     MOVE INREC TO OUTREC
19     LIST DIST (DISTRIBUTOR) FILM GROSS GUARANTY (GUARANTEE)
20     WRITEMEM OUTPDS
21     IF NEWSTAT EQ 'N'
22         PRINT 'I/O ERROR'
```

Figure 629 Example 7 Add New Members to a PDS (Page 1 of 2)

```

23         STOP
24     ENDIF
25     READ TRANSIN
26     ENDDO

27     MOVE SAVECODE TO NEWDATA
28     WRITEDIR OUTPDS ADD
29     IF NEWSTAT EQ 'N'
30         LIST SAVECODE 'IS ALREADY ON FILE'
31     STOP
32     ENDIF
33     ENDDO
34     STOP

35     T1 'VISION:RESULTS PDS EXAMPLE 7' WITH 3 AFTER

```

Figure 629 Example 7 Add New Members to a PDS (Page 2 of 2)

```

VISION:RESULTS PDS EXAMPLE 7

```

DISTRIBUTOR	FILM	GROSS	GUARANTEE
LION MOVIES INCORPORATED	BRIDE OF FRANKENSTEIN	15,000	1,250
LION MOVIES INCORPORATED	THE THING	23,000	2,000
OLD AND GREAT FILMS	CASABLANCA	54,900	4,900
OLD AND GREAT FILMS	2001 A SPACE ODYSSEY	39,040	2,750
OLD AND GREAT FILMS	GONE WITH THE WIND	19,800	550
ACME CINEMA	PINOCCHIO	87,300	8,700
ACME CINEMA	MARY POPPINS	48,800	3,870

Figure 630 [Example 7](#) Output—Add New Members to a PDS

PDS Directory Format

Note: Regarding the relative address (TTR) of the first block, all addresses are relative to the first block of the data set. TT = relative track from beginning of data set. R = block number of that track.

BYTE	LENGTH (IN BYTES)	
2	8	Member or alias name
10	3	Relative address (TTR) of first block of member
13	1	Indicators Bit 0 - If 1, name is an alias. Bits 1=2 - Number of TTRs, bytes. 14-35 Bits 3=7 - Length of user data field in half words.
14	3	TTR of first block of text.
17	1	Not used.
18	3	TTR of node list or scatter table for modules in scatter load or overlay structure only.
21	1	Number of entries in node list for modules in overlay structure only.
22	2	Attributes of load module: Byte 1 Bit 0=1 - Re-enterable. Bit 1=1 - Reusable. Bit 2=1 - Overlay structure. Bit 3=1 - Testran. Bit 4=1 - Loadable only. Bit 5=1 - Scatter format. Bit 6=1 - Executable. Bit 7=1 - No relocation dictionary - one block of text Bit 7=0 - Multiple records in module Byte 2 Bit 0=1 - F level linkage editor only Bit 0=0 - All levels of linkage editor Bit 1=1 - Origin of first block of text=0 Bit 1=0 - Origin of first block of text=0 Bit 2=1 - EPA is zero Bit 3=1 - No relocation dictionary items Bit 4=1 - Module cannot be processed by linkage editor Bit 5=1 - Module contains testran cards Bit 6=1 - Module created by level F linkage editor Bit 7=1 - Refreshable
24	3	Contiguous memory requirements
27	2	Block length of first text block
29	3	EPA - entry point address
32	3	Linkage editor assigned origin of first test block
If module has an alias name and is reusable or re-enterable		
35	2	Entry point

Figure 631 PDS Directory Format (Page 1 of 2)

BYTE	LENGTH (IN BYTES)	
38	8	Member name when bytes 2-9 contain an alias name
If load module is in scatter format		
35	2	Size of scatter list
37	2	Bytes in translation table
39	2	External symbol directory identification of the CSECT to which first block of text belongs
41	2	ESDID of CSECT containing entry point
Scatter format with re-enterable and reusable attributes		
43	3	Entry point
46	8	Member name if name in bytes 2-9 is an alias

Figure 631 PDS Directory Format (Page 2 of 2)

Source Statement Library Support

You can directly access books in VSE source statement libraries (SSL), reading them as files. They can be selectively extracted, interrogated, and printed from any automatically searched SSL. Updating books is not supported.

Input to VISION:Results

The SSL FILE statement has the same format as the VISION:Results FILE statement, with the addition of the keywords SSL and BOOK. The complete format of the SSL FILE statement is:

```
FILE filename SSL
BOOK dataname STATUS dataname
[COUNT dataname]
```

Figure 632 SSL FILE Statement

FILE

Required. The keyword FILE identifies this statement as a file definition. The keyword SSL defines the file as a source statement library. The file name must be a 1- to 7-character name used to refer to the file in procedure logic. This name bears no relationship to the file name used in the DLBL statement in your JCL. Standard VSE system file names must be used to define a system or private SSL.

BOOK Dataname

Required. Dataname is a self-defining data name, 2 to 10 characters long, that names a field used to hold up to a 10-character book name (includes the member type).

STATUS Dataname

Required. Dataname is a self-defining data name, 2 to 10 characters long, that contains one of the following values after a READBOOK command is executed:

Y Successful read.
N Book was not found.
E End-of-book.

COUNT Dataname

Optional. Dataname is a self-defining data name, 2 to 10 characters long, that contains a running total of the number of successful reads against the file. This is a 5-byte packed decimal field that you can reset at any time.

Defining the File

In the following example, you define a file (SRCELIB) and a record area (BOOKREC):

```
FILE SRCELIB SSL BOOK BOOKNAM STATUS LIBSTAT  
COUNT INCNT  
BOOKREC 80
```

Figure 633 Defining a File and a Record Area

A number of files can be defined in a single request. Two or more SSL files are required if you need access to two or more books concurrently.

Reading a Book

Reading the first record of a book is accomplished by putting the name of the book in the BOOK dataname and issuing a READBOOK command:

```
MOVE 'C.MASTREC' TO BOOKNAM  
READBOOK SRCELIB
```

Figure 634 Reading the First Record of a Book

The STATUS dataname should be checked, following the READBOOK command, to ensure that the book was found and a record read. The first book record is returned to the 80-character record area (BOOKREC) that follows the FILE statement.

Subsequent READBOOK commands obtain additional records from the book until end-of-book is returned or you change the book name. Each time a new book name is supplied, the source statement libraries are searched and the first record of that book is returned, if found.

The book name must be in one of the following formats:

- sublib.bookname
- bookname

In the second case, VISION:Results adds the member type prefix D. to the supplied book name before searching.

SSL Examples

The following examples illustrate using the SSL function.

Example 1 List Specific Books

This program lists specific books whose names are supplied following the program.

```
FILE SYSIPT CARD STATUS CARDSTAT
  MEMNAME 10
FILE LIBFILE SSL BOOK BOOKNAME STATUS LIBSTAT
  BOOKREC 80

CONTROL MEMNAME

IF CARDSTAT EQ 'E' STOP ENDIF          ; END OF JOB
MOVE MEMNAME TO BOOKNAME
READ LIBFILE
IF LIBSTAT EQ 'N' REJECT ENDIF          ; BOOK NOT FOUND
PRTBOOK:
  LIST BOOKREC AT 1
  READ LIBFILE
  IF LIBSTAT EQ 'Y' GOTO PRTBOOK ENDIF

ON CHANGE IN MEMNAME
  MOVE 66 TO DYLLINE                    ; FORCE NEW PAGE
  MOVE 0 TO DYLPAGE                     ; RESET PAGE NO.

T1 'BOOKNAME: ' FIXED WITH 3 AFTER
T1+12 MEMNAME
T1+39 DYLETIME
T1+51 DYLDATAPAG
FIN
/*(input statements)
```

Figure 635 [Example 1](#) List Specific Books

Example 2 Use a Source Statement Library

This program uses a table of labor rates stored in a source statement library to calculate billing amounts by job number within customer number, and then it prints a report.

```
OPTION TSIZE 10K DATA 'D.TABLE01'

FILE TABLEFL SSL BOOK TBLNAME STATUS TBFLSTAT
  TBFLREC 50

TABLE TDATA F 38 ENTRY TNTRY KEYLOC 1 KEYLEN 3 STATUS TSTAT
  TLABGRADE 3
  TLABDESC 30 (LABDESC)
  TLABRATE 5 NU 3
  TBREC 38 1

FILE WIPFILE ESDS F 75
  WJOBNO 3 PD (JOBNO)
  WCUSTNO 4 PD (CUSTNO)
  WHOURS 3 40 PD 1 B (HOURS)
  WLABGRADE 3 51 (LBG)

WORKAREA
  BILLAMT 4 PD 2 A

T1 'INTERIM BILLING REPORT' WITH 2 AFTER
T1+90 DYLDATAPAG

CONTROL WJOBNO WCUSTNO

SORT WIPFILE USING WCUSTNO WJOBNO
ON ONE
  MOVE DYLPARM TO TBLNAME ;SET TABLE NAME TO RETRIEVE
  LOADTB: ;LOAD TABLE FILE INTO CORE
  READ TABLEFL
  IF TBFLSTAT EQ 'N'
    PRINT 'BAD TABLE NAME'
    MOVE 301 TO DYLRRETURN
    STOP ENDIF
  IF TBFLSTAT EQ 'E'
    GOTO ENDLOAD ENDIF
  IF TSTAT NE 'Y'
    PRINT 'NO MORE MEMORY'
    MOVE 302 TO DYLRRETURN
    STOP ENDIF
  GOTO LOADTB
ENDLOAD:
ENDONE

BINSEARCH TDATA WLABGRADE ;FIND RELATED TABLE ENTRY
  IF TSTAT NE 'Y'
    MOVE '** NOT IN TABLE **' TO TLABDESC
    MOVE 0 TO TLABRATE
    MOVE 303 TO DYLRRETURN ENDIF
  BILLAMT = WHOURS * TLABRATE ROUNDED ;CALCULATE BILLING AMT
  LIST SUPPRESS (WCUSTNO WJOBNO)
  TLABDESC WLABGRADE WHOURS BILLAMT

ON CHANGE IN WJOBNO
  LIST SUM BILLAMT WITH 2 AFTER
ON CHANGE IN WCUSTNO
  LIST SUM BILLAMT '**' WITH EJECT AFTER
ON FINAL
  LIST 'FINAL TOTAL' AT WLABGRADE SUM BILLAMT
```

Figure 636 [Example 2](#) Use a Source Statement Library

Chapter 24: Using MATCH and MERGE

You can use special MATCH and MERGE statements with VISION:Results that provide automatic file advancement, automatic processing of multiple keys on multiple files, and easy access to file information. It makes it fast and convenient for you to sequentially access multiple files simultaneously, processing the data as you go.

A single file can be checked for duplicate records, or records from two or more files can be matched and merged during processing. You can add VISION:Results logic for additional processing and data selection.

The rules for file advancement and other aspects of the MATCH and MERGE feature are explained in this chapter.

MATCH and MERGE consists of two modules: DY282MAT and DY282MER, through which you use the MATCH and MERGE statements or functions. These statements make it possible to accomplish almost any conceivable match or merge task.

Using the MATCH and MERGE feature, it is easy to simultaneously process data from multiple files. Two or more files can be processed sequentially using this facility. As the files are matched (or merged), your program logic is free to process the selected records.

The MATCH facility provides two generalized functions. The first (and most common) is as a general facility for MASTER/TRANS updating. One or more transaction files can be used to update a given master file. The second function compares two or more files to see if they are identical. You can use the MATCH facility to do any necessary file processing between two or more files based on keys.

The MERGE facility provides for generalized MERGE logic between two or more files based on keys. It provides for a MERGE function with logic capabilities.

Special conditional expressions are available in the MATCH and MERGE facilities to help determine the exact relationship between files as they are being matched and merged. The files are advanced according to the rules explained in [Match Conditions in the Procedure Logic on page 643](#) and [Merge Conditions in the Procedure Logic on page 659](#).

The MATCH and MERGE facility gives you a set of tools for automatic processing of multiple input files using the supplied algorithms.

It is possible that you need your own custom MATCH or MERGE subroutine, replacing the built-in DY282MAT and DY282MER modules.

You may not want to adhere to the rules or requirements set up by DY282MAT or DY282MER. For this reason, [Writing Your Own MATCH Subroutine on page 664](#) and [Writing Your Own MERGE Subroutine on page 679](#) have been included. [Writing Your Own MATCH Subroutine on page 664](#) shows you how to handle your own MATCH subroutines. [Writing Your Own MERGE Subroutine on page 679](#) shows you how to handle your own MERGE subroutines. If you are not interested in writing your own custom MATCH or MERGE subroutine, see [MATCH and MERGE Features on page 640](#).

The custom subroutines you write to replace DY282MAT and DY282MER must be written in Assembler, adhere to standard Assembler programming conventions, and be assembled and linked using standard procedures such as those outlined in [Chapter 13: Using the CALL Command](#).

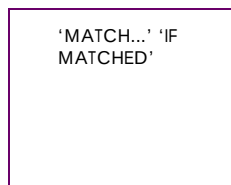
The subroutines must maintain the parameters, flags, and function codes described in [Writing Your Own MATCH Subroutine on page 664](#) and [Writing Your Own MERGE Subroutine on page 679](#). Your subroutine must use the same parameters, flags, function and status codes as DY282MAT, and your subroutine must set the 'read ahead' flag and status code as DY282MAT would. All parameters, including flags and function and status codes passed to and from DY282MAT and DY282MER, are explained in [Parms Passed to DY282MAT on page 672](#) and [Parms Passed to DY282MER on page 682](#).

[Writing Your Own MATCH Subroutine on page 664](#) and [Writing Your Own MERGE Subroutine on page 679](#) each present the process by which DY282MAT and DY282MER accomplish their respective MATCH and MERGE function along with an accompanying flowchart. Each parameter that you must pass, each flag that you must set, and each function and status code that you must specify are described in detail so that you know exactly what your Assembler subroutine must do to replace VISION:Results' corresponding modules. There is also an example of a MATCH subroutine to replace DY282MAT (see [Writing Your Own MATCH Subroutine on page 664](#)) and a MERGE subroutine to replace DY282MER (see [Writing Your Own MERGE Subroutine on page 679](#)), each written in Assembler.

General Flow of Execution Diagram

The following diagram illustrates the overall flow of the VISION:Results MATCH and MERGE feature. After one automatic cycle is complete, the process continues until all files are at the end of file.

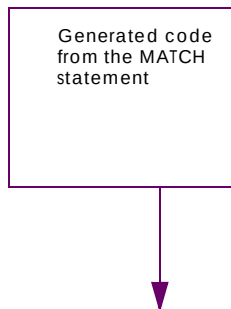
Block 1



This is your VISION:Results program using MATCH and MERGE statements and any additional VISION:Results code required by you. A first-cycle call to DY282MAT is made at the initial startup of the program to set the initial Read Ahead flags for the first read from each file. For further explanation of a first-cycle call, see [DY282MAT Invocation on page 664](#).

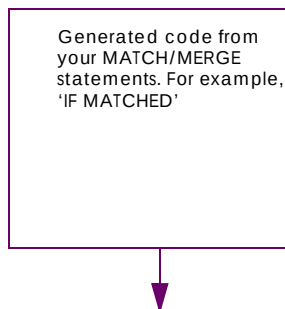
A record is read from the files whose Read Ahead flag contains a Y. This was set by DY282MAT during the new-cycle call in the preceding cycle or the first time only by the first-cycle call. All reads occur between Blocks 1 and 2.

Block 2



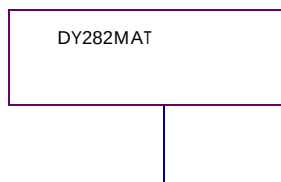
VISION:Results code is generated from a single required MATCH statement in your VISION:Results program. This is the new-cycle call to DY282MAT. Its main function is to set the Read Ahead flags for the next cycle. An ALL MATCHED flag is set if all keys on all files match. Other internal flags to keep track of DUPLICATES are set also. For further explanation of a new-cycle call, see [DY282MAT Invocation on page 664](#).

Block 3



This VISION:Results code is generated from the rest of your MATCH and MERGE statements (for example, IF MATCHED ..., IF FIRSTDUP ..., IF LASTDUP ..., and so on). This code is referred to as the additional USER CODE at the bottom of an expanded detail flowchart. This flowchart can be found in [Generated from Your Special MATCH Logic Statements on page 667](#). DY282MAT can be called more than once by this routine. The status code is set as a result of each of these calls. For further explanation, see [DY282MAT Invocation on page 664](#).

Block 4



DY282MAT is the routine that does all the matching of record keys from all files. DY282MAT determines which file is to be read next and sets a Read Ahead flag accordingly. DY282MAT also sets a status code flag according to the function code received and the result of record key tests.

MATCH and MERGE Features

You can process data simultaneously from multiple files using the MATCH and MERGE feature. Two or more files can be processed sequentially using this facility. As the files are matched or merged, you can process the selected records in your processing logic in the usual way.

Additionally, special conditional expressions are available in these operations to help you determine the precise relationships between the files as they are being matched or merged.

The files are advanced in MATCH and MERGE according to rules that are explained in [Match Conditions in the Procedure Logic on page 643](#) and [Merge Conditions in the Procedure Logic on page 659](#), respectively.

Using the MATCH operation, you can:

- MATCH two or more files sequentially processed using up to nine keys each.
- Use special IF statements to check for a match on all files, or a match on any two files.
- Check for duplicates on any file, first duplicate and last duplicate on a file, and for whether a file was last advanced during the automatic MATCH cycle.
- Use the special IF conditions in the usual way to form complex conditional expressions.
- Process the records from the MATCH in the usual way in any request.

Using the MERGE operation, you can:

- MERGE (interleave) two or more files sequentially processed using up to nine keys each.
- Use a special IF statement to check for replicas in the merged file area.
- Use the special IF statement with any other conditional expressions to form complex conditional expressions in the usual way.
- Process the records from the MERGE in the usual way in any request.

The MATCH and MERGE operations give you a set of tools for automatically processing multiple input files using either the supplied algorithms or your own.

Syntax for the MATCH Operation

All files referenced in the MATCH must be defined using the usual FILE and field definitions in preceding statements. The format of the MATCH statement is:

```
MATCH filename1 KEY1 dataname [KEY2 dataname KEY3 dataname...KEY9 dataname]
      filename2 KEY1 dataname [KEY2 dataname KEY3 dataname...KEY9 dataname]
      [filenamen KEY1 dataname {KEY2 dataname KEY3 dataname...KEY9 dataname}]
      [SETREAD {MASTERUP|COMPAREN|ORIGINAL}] '
      [USING {DY282MAT|Subroutine}]
```

One or more special IF statements here

```
(IF [ADVANCED | FIRSTDUP | LASTDUP | MATCHED | DUPLICATE])
```

Figure 637 Format of the MATCH Statement

MATCH	Required. It declares the application as a MATCH operation.
filename1	Required. The file name is the name of the first file designated for the MATCH operation. The file must be defined in preceding statements.
KEY1 dataname	Required. The data name is used as the first key field for the MATCH operation. filename1 must be in ascending sequence by this key.
KEY2 dataname, KEY3 dataname, through KEY9 dataname	Optional. These data names identify the other key fields to be used for the MATCH operation. If specified, filename1 must be in ascending sequence by these keys.
filename2	Required. This file name is the name of the second file designated for the MATCH operation. The file must be defined in preceding statements.
KEY1 dataname	Required. The data name is used as the first key field for the MATCH operation. filename2 must be in ascending sequence by this key.
KEY2 dataname, KEY3 dataname, through KEY9 dataname	Optional. These data names identify the other key fields for the second file for the MATCH operation. If specified, filename2 must be in ascending sequence by these keys.
filenamen	Optional. These file names identify any number of additional files to be included in the MATCH operation.
KEY1 dataname	Required if a file name has been specified as filenamen. The filenamen must be in ascending sequence by this key.
KEY2 dataname, KEY3 dataname, through KEY9 dataname	Optional. These data names identify the other key fields for the file named in filenamen. If specified, the filenamen must be in ascending sequence by these keys.

SETREAD MASTERUP, COMPAREN, and ORIGINAL

SETREAD MASTERUP is the default. It indicates the type of file advancement to use for this MATCH run.

MASTERUP is used for master/transaction files type of updating. It advances as many records as possible during each cycle. It locks the first of a set of duplicate masters if a matching set of duplicate transactions is available to update.

SETREAD COMPAREN is used for comparing two or more files to see if they are identical. It also advances as many records as possible during each cycle. No locking is done.

SETREAD ORIGINAL is the original MATCH file advancement logic. It advances one record during each cycle according to its tie-breaking logic. For information about how tie breaking is accomplished, see [Match Conditions in the Procedure Logic on page 643](#). It can be used in place of either of the above two functions, but its file advancement is more complex. Study it thoroughly before you attempt to write a MATCH program using it.

USING DY282MAT

Optional. The default is USING DY282MAT. This indicates the type of MATCH operation to use. DY282MAT allows all the special IF conditions during the match to be used. You can also specify the name of your own subroutine to be used for the matching.

Recommendations and Requirements for Matching

Specify at least two files during a MATCH, with the following provisions:

- Each file in the MATCH must have at least one key specified.
- Every file in the MATCH must have the same number of keys specified.
- Each respective key in each file specified in the MATCH must be of the same type and length and have the same number of decimal positions. For example, KEY1 on file1 and KEY1 on file2 must be of the same type and length and have the same number of decimal positions.
- Every file in the MATCH must be previously defined in a file definition.
- For all files in a MATCH, the record format and record length entries must be specified in their respective file definitions.
- For all files in a MATCH, we recommend that the files be in ascending sequence by their respective keys. The files are not sequence checked.
- SORT is not allowed during a request with a MATCH. However, the files can be sorted in a prior request and passed to the current MATCH request.
- MATCH is not allowed with OPTION STRUCTURED.

- An output file cannot be OUTPUT FROM a file that is referenced in a MATCH statement.
- The MATCH statement must be coded before any other processing statements.
- The ON END OF INPUT statement cannot be used in a MATCH request.

Match Conditions in the Procedure Logic

The key to understanding MATCH logic is to understand how each of the file advancement techniques works. VISION:Results automatically reads the files to be matched based on the keyword chosen with the SETREAD option (where MASTERUP is the default). The records are then presented to your logic for your processing. Special IF statements are available for you to check on the results of the matches during the automatic cycle. These special IF statements are explained in detail later in this section.

All file advancement techniques are based on the concept of current and next keys. Current keys are on the current record in storage. The MATCH logic has access to the next record in storage also (due to a look-ahead read). The actual file positioning is transparent to you. See Examples 1, 3, and 4 in [MATCH Examples on page 652](#).

SETREAD MASTERUP

With SETREAD MASTERUP (Master or Transaction updating), the files are advanced as follows: The next keys are compared. The file or files with the next lowest keys are read. The exception to this is if multiple duplicate masters exist and multiple duplicate transactions for that master exist, the first master is locked in place until all the duplicate transactions are read (allowing you to make updates with the duplicate transaction against the first duplicate master), then the remaining duplicate masters are read.

SETREAD COMPAREN

With SETREAD COMPAREN (two or more files compared to see if they are identical), the files are advanced exactly as SETREAD MASTERUP, except no locking occurs. This is because the reason for the comparison is to see if the given files are identical. It is your responsibility to check for duplicates in unusual cases.

SETREAD ORIGINAL

With SETREAD ORIGINAL (the original VISION:Results MATCH file advancement technique), the files are advanced one record at a time. First, the current keys are compared. Then, the record with the lowest keys is read. If all the current keys are equal, the next keys are compared. The record with the next lowest keys is read next. If all the next keys are equal, the file with the highest precedence is read (the first file specified in the MATCH statement/this is its tie-breaking logic).

DY282MAT

In the following example, assume three files are being matched:

```
FILE FILEA FB 80
  RECA 80
  KEY1A 1 1
FILE FILEB FB 80
  RECB 80
  KEY1B 1 1
FILE FILEC FB 80
  RECC 80
  KEY1C 1 1

MATCH  FILEA KEY1 KEY1A
      FILEB KEY1 KEY1B
      FILEC KEY1 KEY1C
      SETREAD MASTERUP
```

Your procedure logic goes here

Figure 638 DY282MAT

The keys on the three files are as follows (duplicate keys on a file are shown here with an apostrophe '):

FILEA	FILEB	FILEC
2	1	2
2'	2	2'
2'	2'	4
3	2'	7
5	3	7'
6	4	9
7	6	
8	6'	

The records input to the procedure logic at the beginning of each cycle are as follows:

Note: The highlighted (**bold**) values indicate the next-record read from that file.

CYCLE NUMBER	FILEA KEY1	FILEB KEY1	FILEC KEY1	
1	low-values	1	low-values	
2	2	2	2	
3	2	2'	2'	← FILEA 'locked'
4	2	2"	2'	← FILEA 'locked'
5	2'	2'	2'	
6	2"	2'	2'	
7	3	3	2'	
8	3	4	4	
9	5	4	4	
10	6	6	4	
11	6	6'	4	
12	7	6'	7	
13	7	6'	7'	
14	8	6'	7'	
15	8	6'	9	

You can understand the logic by following the file advancement through some of the cycles. The files are advanced by comparing the next lowest keys and advancing the files accordingly.

- Cycle 1 FILEB has the next lowest key of 1, so it is read first. FILEA and FILEC initially have X'00' in their current key buffer. If the keys were numeric fields, they are numeric zero based on type.
- Cycle 2 All three files MATCH, so they are all advanced.
- Cycle 3 All three files MATCH, but there are duplicate transactions, so the master (FILEA) is not advanced to allow updating against the first duplicate master for your benefit. FILEA is locked until all duplicates from FILEB and FILEC are read.
- Cycle 4 FILEA and FILEB MATCH. Again, FILEA is locked.
- Cycle 5 All duplicate transactions have been read, so a duplicate master FILEA is read.
- Cycle 6 The last duplicate master FILEA is read.
- Cycle 7 FILEA and FILEB MATCH, so they are both read.
- and so on

The records available to your logic at the beginning of each cycle are as follows for COMPAREN:

CYCLE NUMBER	FILEA KEY1	FILEB KEY1	FILEC KEY1
1	low-values	1	low-values
2	2	2	2
3	2'	2'	2'
4	2'	2'	2'
5	3	3	2'
6	3	4	4
7	5	4	4
8	6	6	4
9	6	6'	4
10	7	6'	7
11	7	6'	7'
12	8	6'	7'
13	8	6'	9

The file advancement for COMPAREN is identical to MASTERUP, except it does not lock the first duplicate master in the event of duplicate transactions. It is only comparing to see if the files are identical.

The records available to your logic at the beginning of each cycle are as follows for ORIGINAL (one record is read at a time):

CYCLE NUMBER	FILEA KEY1	FILEB KEY1	FILEC KEY1
1	low-values	1	low-values
2	2	1	low-values
3	2	1	2
4	2	2	2
5	2'	2	2
6	2'	2	2
7	2'	2'	2
8	2'	2'	2
9	2'	2'	2'
10	3	2'	2'
11	3	3	2'
12	3	3	4
13	3	4	4
14	5	4	4
15	5	6	4
16	5	6	7
17	6	6	7
18	6	6'	7
19	7	6'	7
20	7	6'	7'
21	8	6'	7'
22	8	6'	9

The file advancement for ORIGINAL uses its own tie-breaking logic for advancing files as detailed previously. If you use this technique for master/transaction updating, it is recommended to use two files only and disallow duplicate masters; otherwise, use MASTERUP. If this technique is used for comparing, it is recommended to use two files only; otherwise, use COMPAREN. It works for more than two files as above, but the logical relationships between the files become increasingly complex.

IF ADVANCED file1

The special IF ADVANCED file1 statement in your procedure logic allows you to test a given file during the automatic MATCH cycle for whether a new record was read from the specified file on this cycle (the current record is a new record from the file). If so, the condition is true; if not, the condition is false.

The format of the IF ADVANCED file1 statement is:

```
IF [NOT] ADVANCED filename [AND|OR condition . . .] imperative . . . ENDIF
```

Figure 639 Format of the IF ADVANCED file1 Statement

For example, the following IF statements are valid:

```
IF ADVANCED FILEA
  GOTO WRITOUT ENDIF
IF NOT ADVANCED FILEA
  GOTO LABEL ENDIF
```

The IF ADVANCED file1 statement can be used with any other conditional expressions using the AND and OR connectors in the usual way. This IF statement is like any other IF statement in VISION:Results and can be negated by using an IF NOT ADVANCED file1 statement. The IF ADVANCED file1 statement can be used with any of the other special IF statements to test for combinations of conditions.

The following example illustrates the points in the MATCH cycle when the IF ADVANCED file1 statement is true. Suppose the following three files are used in the MATCH operation:

FILEA	FILEB	FILEC
1	1	1
2	3	2
2'	3'	3
5	4	3'

The following example uses the MASTERUP file advancement technique. The records output to the procedure logic are tested with an IF ADVANCED FILEA statement for a true or false condition:

FILEA	FILEB	FILEC	IF ADVANCED FILEA test
1	1	1	True
2	1	2	True
2'	1	2	True
2'	3	3	False
2'	3'	3'	False
2'	4	3'	False
5	4	3'	True

The IF ADVANCED file1 condition is true only if the current record is a new record from the specified file during the automatic MATCH cycle. In the above example, the keys on FILEA 1, 2, 2', and 5 represent the new records advanced during the cycle at the points highlighted in **bold** type.

IF MATCHED

The special IF MATCHED statement in your procedure logic allows you to test whether all keys across all files are equal. If they are, the condition is true; if not, the condition is false.

The format of the IF MATCHED statement is:

```
IF [NOT] MATCHED [AND|OR condition . . .] imperative . . . ENDIF
```

Figure 640 Format of the IF MATCHED Statement

For example, the following IF statements are valid:

```
IF MATCHED AND KEY1A EQ 123
  LIST RECA ENDIF
IF NOT MATCHED
  GOTO LABEL ENDIF
```

The IF MATCHED statement can be used with any other conditional expressions using the AND and OR connectors in the usual way. This IF statement is like any other IF statement in VISION:Results. It can be negated by saying IF NOT MATCHED. The IF MATCHED statement can be used with any of the other special IF statements to test for combinations of conditions.

The following example illustrates the points in the MATCH cycle when the IF MATCHED statement is true. Suppose the following three files are used in the MATCH operation:

FILEA	FILEB	FILEC
1	1	1
2	3	2
2'	3'	3
5	4	3'
	5	5

The following example uses the MASTERUP file advancement technique. The records input to the procedure logic are tested with an IF MATCHED statement for a true or false condition:

FILEA	FILEB	FILEC	IF MATCHED test
1	1	1	True
2	1	2	False
2'	1	2	False
2'	3	3	False
2'	3'	3'	False
2'	4	3'	False
5	5	5	True

The IF MATCHED condition is true only when all the keys match across all files.

IF MATCHED file1 file2

The special IF MATCHED file1 file2 statement in your procedure logic allows you to test whether all keys across any two specified files are equal. If they are, the condition is true; if not, the condition is false.

The format of the IF MATCHED file1 file2 statement is:

```
IF [NOT] MATCHED file1 file2 [AND|OR condition . . .] imperative . . . ENDIF
```

Figure 641 Format of the IF MATCHED file1 file2 Statement

For example, the following IF statements are valid:

```
IF MATCHED FILEA FILEB AND (KEY1A EQ 123 AND KEY2B EQ 123)
  LIST RECA RECB ENDIF
IF NOT MATCHED FILEA FILEB
  GOTO LABEL ENDIF
```

The IF MATCHED file1 file2 statement can be used with any other conditional expressions using the AND and OR connectors in the usual way. This IF statement is like any other IF statement in VISION:Results. It can be negated by saying IF NOT MATCHED file1 file2. The IF MATCHED file1 file2 statement can be used with any of the other special IF statements to test for combinations of conditions.

The following example illustrates the points in the MATCH cycle when the IF MATCHED file1 file2 statement is true. Suppose the following three files are used in the MATCH operation:

FILEA	FILEB	FILEC
1	1	1
2	1'	2
2'	3	2'
4	4	5
9		6

The following example uses the MASTERUP file advancement technique. The records input to the procedure logic are tested with an IF MATCHED FILEA FILEB and an IF MATCHED FILEA FILEC for a true or false condition:

FILEA	FILEB	FILEC	IF MATCHED FILEA IF MATCHED FILEA	FILEB FILEB	FILEC FILEC
1	1	1		True	True
1	1'	1		True	True
2	1'	2	False	False	True
2	1'	2'	False	False	True
2'	1'	2'	False	False	True
2'	3	2'	False	False	True
4	4	2'	True	False	False
4	4	5	True	False	False
4	4	6	True	False	False
9	4	6	False	False	False

The IF MATCHED file1 file2 condition is true only when all the keys for file1 and file2 match. The IF MATCHED file1 file2 statement works with the IF ADVANCED|NOT ADVANCED file2 statement with MASTERUP processing to either update the master file with the transaction or reject the transaction as invalid.

IF DUPLICATE file1

The special IF DUPLICATE file1 statement in your procedure logic allows you to test a given file during the automatic MATCH cycle for whether the current key is equal to the previous key or the current key is equal to the next key. If the keys are equal, the condition is true; if not, the condition is false.

During the cycle, the current record is the one that is available to you during the processing logic, the previous record is the one prior to the record being processed, and the next record is a look-ahead record to become the next current record for the file at the appropriate time according to the rules of the MATCH file advancement technique you are using.

The format of the IF DUPLICATE file1 statement is:

```
IF [NOT] DUPLICATE file1 [AND|OR condition . . .] imperative . . . ENDIF
```

Figure 642 Format of the IF DUPLICATE file1 Statement

For example, the following IF statements are valid:

```
IF DUPLICATE FILEA
  LIST RECA ENDIF
IF NOT DUPLICATE FILEA
  GOTO LABEL ENDIF
```

The IF DUPLICATE file1 statement can be used with any other conditional expressions using the AND and OR connectors in the usual way. This IF statement is like any other IF statement in VISION:Results. It can be negated by saying IF NOT DUPLICATE file1. The IF DUPLICATE file1 statement can be used with any other special IF statements to test for combinations of conditions.

The following example illustrates the points in the MATCH cycle when the IF DUPLICATE file1 statement is true. Suppose the following three files are used in the MATCH operation:

FILEA	FILEB	FILEC
1	1	1
2	3	2
2'	3'	3
5	4	3'

The following example uses the MASTERUP file advancement technique. The records input to the procedure logic are tested with an IF DUPLICATE FILEA statement for a true or false condition:

FILEA	FILEB	FILEC	IF DUPLICATE FILEA
1	1	1	False
2	1	2	True
2'	1	2	True
2'	3	3	True
2'	3'	3'	True
2'	4	3'	True
5	4	3'	False

In the above example, the keys 2 and 2' are duplicates; therefore, the test IF DUPLICATE FILEA is always true whenever those two keys are present in the automatic cycle (either as the current and next keys or as the current and previous keys).

IF FIRSTDUP file1

The special IF FIRSTDUP file1 statement in your procedure logic allows you to test a given file during the automatic MATCH cycle for whether the current key is not equal to the previous key but is equal to the next key. If so, the condition is true; if not, the condition is false. This is a test for the first duplicate of two or more duplicates on a file.

The format of the IF FIRSTDUP file1 statement is:

```
IF [NOT] FIRSTDUP file1 [AND|OR condition . . .] imperative . . . ENDIF
```

Figure 643 Format of the IF FIRSTDUP file1 Statement

For example, the following IF statements are valid:

```
IF FIRSTDUP FILEA
  LIST RECA ENDIF
IF NOT FIRSTDUP FILEA
  GOTO LABEL ENDIF
```

The IF FIRSTDUP file1 statement can be used with any other conditional expressions using the AND and OR connectors in the usual way. This IF statement is like any other IF statement in VISION:Results. It can be negated by saying IF NOT FIRSTDUP file1. The IF FIRSTDUP file1 statement can be used with any other special IF statements to test for combinations of conditions.

The following example illustrates the points in the MATCH cycle when the IF FIRSTDUP file1 statement is true. The following three files are used in the MATCH operation:

FILEA	FILEB	FILEC
1	1	1
2	3	2
2'	3'	3
5	4	3'

The following example uses the MASTERUP file advancement technique. The records input to the procedure logic are tested with an IF FIRSTDUP FILEB statement for a true or false condition:

FILEA	FILEB	FILEC	IF FIRSTDUP FILEB
1	1	1	False
2	1	2	False
2'	1	2	False
2'	3	3	True
2'	3'	3'	False
2'	4	3'	False
5	4	3'	False

The IF FIRSTDUP file1 condition is true only if the current key is not equal to the previous key and the current key is equal to the next key during the automatic MATCH cycle. In the above example, the keys 3 and 3' are duplicates (of which 3 is the first duplicate), so that the test IF FIRSTDUP FILEB is always true whenever the key 3 is present in the automatic cycle.

IF LASTDUP file1

The special IF LASTDUP file1 statement in your procedure logic allows you to test a given file during the automatic MATCH cycle for whether the current key is equal to the previous key and not equal to the next key. If so, the condition is true; if not, the condition is false. This is a test for the last duplicate of two or more duplicates on a file.

The format of the IF LASTDUP file1 statement is:

```
IF [NOT] LASTDUP file1 [AND|OR condition . . .] imperative . . . ENDIF
```

Figure 644 Format of the IF LASTDUP file1 Statement

For example, the following IF statements are valid:

```
IF LASTDUP FILEA
  LIST RECA ENDIF
IF NOT LASTDUP FILEA
  GOTO LABEL ENDIF
```

The IF LASTDUP file1 statement can be used with any other conditional expressions using the AND and OR connectors in the usual way. This IF statement is like any other IF statement in VISION:Results. It can be negated by saying IF NOT LASTDUP file1. The IF LASTDUP file1 statement can be used with any other special IF statements to test for combinations of conditions.

The following example illustrates the points in the MATCH cycle when the IF LASTDUP file1 statement is true. Suppose the following three files are used in the MATCH operation:

FILEA	FILEB	FILEC
1	1	1
2	3	2
2'	3'	3
5	4	3'

The following example uses the MASTERUP file advancement technique. The records input to the procedure logic are tested with an IF LASTDUP FILEC statement for a true or false condition:

FILEA	FILEB	FILEC	IF LASTDUP FILEC
1	1	1	False
2	1	2	False
2'	1	2	False
2'	3	3	False
2'	3'	3'	True
2'	4	3'	True
5	4	3'	True

The IF LASTDUP file1 condition is true only if the current key is not equal to the next key and the current key is equal to the previous key during the automatic MATCH cycle. In the above example, the keys 3 and 3' are duplicates (of which 3' is the last duplicate), so the test IF LASTDUP FILEC is always true whenever the key 3' is present in the automatic cycle.

MATCH Examples

Example 1 Simple Updating Using MATCH and File Advancement MASTERUP

No checking is done in Example 1 for duplicate masters or duplicate transactions. If they exist, there are duplicates on the output file.

Any transaction records that MATCH master records are considered a complete replacement for the master record, and any new transaction records are considered to be complete new masters. Any time the transaction record is read (IF ADVANCED INTRANS), it is moved to the output master record (this happens regardless of an IF MATCHED or IF NOT MATCHED condition). In any other case, the input master record is moved to the output master record (this happens in an IF ADVANCED INMSTR AND NOT MATCHED condition).

End of file is handled automatically by the MATCH logic. All the records are presented to you during the current cycles. At the beginning of the next cycle, MATCH issues a [STOP](#). If you want to do additional logic after end of file on all files, the ON FINAL statement can be used (implies report writing of at least one record even if all blank).

```

OPTION XREFS PRINTERR
FILE INMSTR FB 80 STATUS MASTSTAT
  MSTRREC 80
  MSTRID 9 1
  MSTRNAME 15 33
  MSTRNUM 5 51 NU
FILE INTRANS FB 80 STATUS TRANSTAT
  TRANREC 80
  TRANID 9 1
  TRANNAME 15 33
  TRANNUM 5 51 NU
FILE OUTMSTR OUTPUT FB 80 FROM OUTMSTR
  OMSTREC 80
  OMSTID 9 1
  OMSTNAME 15 33
  OMSTNUM 5 51 NU
MATCH INMSTR KEY1 MSTRID KEY2 MSTRNUM KEY3 MSTRNAME
      INTRANS KEY1 TRANID KEY2 TRANNUM KEY3 TRANNAME
      SETREAD MASTERUP
IF ADVANCED INTRANS
  MOVE TRANREC TO OMSTREC
ELSE
  MOVE MSTRREC TO OMSTREC
ENDIF
WRITE OUTMSTR

```

Figure 645 [Example 1](#) Simple Updating Using MATCH and File Advancement MASTERUP

Example 2 Traditional Master or Transaction Updating (Default to SETREAD MASTERUP)

All duplicate masters are rejected. Any transactions associated with duplicate masters are rejected.

Transactions can either be updates (if MATCHED) or adds (if NOT MATCHED). Duplicate update transactions are allowed. Duplicate add transactions are rejected.

Input masters that are NOT MATCHED with transactions are written as is to the output master.

A write flag controls the writing of the output master file. At end of file, ON FINAL logic is used to write the last output master. The ON FINAL statement implies the report writer, so a blank line is listed at end of file.

```

REPORT 80 WIDE ASA
OPTION XREFS PRINTERR
FILE INMSTR FB 80 STATUS MASTSTAT
  MSTRREC 80
  MSTRID 9 1
  MSTRNAME 15 33
  MSTRNUM 5 51 NU
  MSTRFLD1 5 56
  MSTRFLD2 5 61
  MSTRFLD3 5 66

```

Figure 646 [Example 2](#) Traditional Master or Transaction Updating (Default to SETREAD MASTERUP) (Page 1 of 3)

```

FILE INTRANS FB 80 STATUS TRANSTAT
  TRANREC      80
  TRANID       9   1
  TRANNAME    15  33
  TRANNUM      5  51 NU
  TRANFLD1     5  56
  TRANFLD2     5  61
  TRANFLD3     5  66
FILE OUTMSTR OUTPUT FB 80 FROM OUTMSTR
  OMSTREC      80
  OMSTID       9   1
  OMSTNAME    15  33
  OMSTNUM      5  51 NU
  OMSTFLD1     5  56
  OMSTFLD2     5  61
  OMSTFLD3     5  66
WORKAREA
  WRITEFLG     1   1 VALUE 'N'
MATCH INMSTR KEY1 MSTRID KEY2 MSTRNUM KEY3 MSTRNAME
  INTRANS KEY1 TRANID KEY2 TRANNUM KEY3 TRANNAME
  * DUPLICATE MASTERS ARE AN ERROR (IF TRANS ADVANCED, ALSO
  * AN ERROR).
  IF ADVANCED INMSTR AND DUPLICATE INMSTR
    MOVE 'MSTR ERROR' TO DYLPRTCOMM
    PRINT INMSTR
    PERFORM CKTRAN TO CKTRANX
    REJECT
  ENDIF
  * IF TRANS ONLY ADVANCED BUT MATCHES WITH DUPLICATE MASTER,
  * ALSO ERROR.
  IF ADVANCED INTRANS AND MATCHED AND DUPLICATE INMSTR
    MOVE 'TRAN ERROR' TO DYLPRTCOMM
    PRINT INTRANS
    REJECT
  ENDIF
  * IF MASTER READ, CHECK FOR WRITE AND MOVE TO OUTPUT FOR NEXT
  * WRITE.
  IF ADVANCED INMSTR
    PERFORM CKWRT TO CKWRTX
    MOVE 'Y' TO WRITEFLG
    MOVE MSTRREC TO OMSTREC
  ENDIF
  * IF MASTER READ AND MATCHED TRANS, UPDATE OUTPUT AREA WITH
  * TRANS.
  IF ADVANCED INTRANS AND MATCHED
    PERFORM UPD TO UPDX
  ENDIF
  * IF TRANS READ AND NO MATCH, MUST BE A NEW ADD.
  IF ADVANCED INTRANS AND NOT MATCHED
    PERFORM ADDIT TO ADDITX
  ENDIF
  ACCEPT
  * IF MASTER READ AND DUPLICATE MASTER, CHECK FOR TRAN ERROR
  * ALSO.
  CKTRAN:
    IF ADVANCED INTRANS
      MOVE 'TRAN ERROR' TO DYLPRTCOMM
      PRINT INTRANS
    ENDIF
  CKTRANX:
    * CHECK WRITE FLAG. IF 'ON', WRITE OUTPUT MASTER.
  CKWRT:
    IF WRITEFLG EQ 'Y'
      MOVE 'N' TO WRITEFLG
      WRITE OUTMSTR
    ENDIF
  CKWRTX:
  * UPDATE TRANSACTION. UPDATE OUTPUT WITH TRANSACTION DATA.
  UPD:
    IF TRANFLD1 NE SPACES
      MOVE TRANFLD1 TO OMSTFLD1
    ENDIF
    IF TRANFLD2 NE SPACES
      MOVE TRANFLD2 TO OMSTFLD2
    ENDIF

```

Figure 646 Example 2 Traditional Master or Transaction Updating
(Default to SETREAD MASTERUP) (Page 2 of 3)

```

        IF TRANFLD3 NE SPACES
          MOVE TRANFLD3 TO OMSTFLD3
        ENDIF
UPDX:
* ADD TRANSACTION. CHECK FOR WRITE AND MOVE TRANSACTION TO
* OUTPUT.
ADDIT:
  IF NOT DUPLICATE INTRANS
    PERFORM CKWRT TO CKWRTX
    MOVE 'Y' TO WRITEFLG
    MOVE TRANREC TO OMSTREC
  ELSE
    MOVE 'TRAN DUPL ' TO DYLPRTCOMM
    PRINT INTRANS
    REJECT
  ENDIF
ADDITX:
* AT EOF ON BOTH FILES, CHECK FOR LAST WRITE OF OUTPUT
* MASTER.
ON FINAL
  PERFORM CKWRT TO CKWRTX
  LIST ' '

```

Figure 646 Example 2 Traditional Master or Transaction Updating
(Default to SETREAD MASTERUP) (Page 3 of 3)

Example 3 Listing Unmatched Records Using File Advancement COMPAREN

If both files are ADVANCED (both must MATCH according to the rules of COMPAREN), accept them as equal. Otherwise, list the unmatched file by determining which one was ADVANCED.

```

OPTION XREFS PRINTERR
FILE LIVEMSTR FB 80 STATUS LIVESTAT
  LIVEREC 80
  LIVEID 9 1 (EMPLOYEE'SSN)
  LIVENAME 15 33 (EMPLOYEE'NAME)
  LIVENUM 5 51 NU (EMPLOYEE'NUMBER)
FILE TESTMSTR FB 80 STATUS TESTSTAT
  TESTREC 80
  TESTID 9 1 (EMPLOYEE'SSN)
  TESTNAME 15 33 (EMPLOYEE'NAME)
  TESTNUM 5 51 NU (EMPLOYEE'NUMBER)
WORKAREA
  WORKORIG 10 (FILE OF'ORIGIN)
MATCH LIVEMSTR KEY1 LIVEID KEY2 LIVENUM KEY3 LIVENAME
  TESTMSTR KEY1 TESTID KEY2 TESTNUM KEY3 TESTNAME
  SETREAD COMPAREN
IF ADVANCED LIVEMSTR AND ADVANCED TESTMSTR
  ACCEPT
ENDIF
IF ADVANCED LIVEMSTR
  MOVE 'LIVE MSTR ' TO WORKORIG
  LIST WORKORIG LIVEID LIVENAME LIVENUM
ELSE
  MOVE 'TEST MSTR ' TO WORKORIG
  LIST WORKORIG TESTID TESTNAME TESTNUM
ENDIF
*
T1 'UNMATCHED RECORDS FOR THE LIVE AND TEST FILES'
  WITH 2 AFTER

```

Figure 647 [Example 3](#) Listing Unmatched Records Using File Advancement
COMPAREN

Example 4 Simple Updating Using File Advancement ORIGINAL

No checking is done in the above example for duplicate masters. If they did exist, there would be duplicates on the output file.

Any matched transaction records are used to update the corresponding master record. Any unmatched transaction records are ignored.

A write flag controls the writing of the output master file. At end of file, ON FINAL logic is writes the last output master. The ON FINAL statement implies the report writer, so a blank line is listed at end of file.

Use SETREAD ORIGINAL only as a last resort because the method of file advancement does not consistently advance files from one file or the other; rather, it is done on a comparison of current keys, then next keys if necessary, and then according to file precedence. This can lead to unnecessarily complex coding solutions to account for all combinations of file advancement. Also, because it advances only one record at a time from any given file, you cannot combine MATCH keywords freely to test logic paths as with the other two methods. For example, for SETREAD ORIGINAL, the test IF ADVANCED INTRANS AND NOT MATCHED would not always indicate a missing master, whereas these tests are absolutely reliable with the other two techniques because of their file advancement techniques.

```
REPORT 80 WIDE ASA
OPTION XREFS PRINTERR
FILE INMSTR FB 80 STATUS MASTSTAT
  MSTRREC      80
  MSTRID       9   1
  MSTRNAME     15  33
  MSTRNUM      5  51 NU
  MSTRFLD1     5  56
  MSTRFLD2     5  61
  MSTRFLD3     5  66
FILE INTRANS FB 80 STATUS TRANSTAT
  TRANREC      80
  TRANID       9   1
  TRANNAME     15  33
  TRANNUM      5  51 NU
  TRANFLD1     5  56
  TRANFLD2     5  61
  TRANFLD3     5  66
FILE OUTMSTR OUTPUT FB 80 FROM OUTMSTR
  OMSTREC      80
  OMSTID       9   1
  OMSTNAME     15  33
  OMSTNUM      5  51 NU
  OMSTFLD1     5  56
  OMSTFLD2     5  61
  OMSTFLD3     5  66
WORKAREA
  WRITEFLG     1   1 VALUE 'N'
MATCH INMSTR KEY1 MSTRID KEY2 MSTRNUM KEY3 MSTRNAME
      INTRANS KEY1 TRANID KEY2 TRANNUM KEY3 TRANNAME
      SETREAD ORIGINAL
IF ADVANCED INMSTR
  PERFORM CKWRT TO CKWRTX
  MOVE 'Y' TO WRITEFLG
  MOVE MSTRREC TO OMSTREC
ENDIF
IF MATCHED
  PERFORM UPD TO UPDX
ENDIF
```

Figure 648 [Example 4](#) Simple Updating Using File Advancement ORIGINAL
(Page 1 of 2)


```

ACCEPT
*
CKWRT:
  IF WRITEFLG EQ 'Y'
    MOVE 'N' TO WRITEFLG
    WRITE OUTMSTR
  ENDIF
CKWRTX:
UPD:
  IF TRANFLD1 NE SPACES
    MOVE TRANFLD1 TO OMSTFLD1
  ENDIF
  IF TRANFLD2 NE SPACES
    MOVE TRANFLD2 TO OMSTFLD2
  ENDIF
  IF TRANFLD3 NE SPACES
    MOVE TRANFLD3 TO OMSTFLD3
  ENDIF
UPDX:
*
ON FINAL
  PERFORM CKWRT TO CKWRTX
LIST ' '

```

Figure 648 [Example 4](#) Simple Updating Using File Advancement ORIGINAL
(Page 2 of 2)

MERGE

The MERGE operation allows you to interleave two or more files that are in sequence using up to nine keys. The output records from the MERGE are available to your request one at a time, in a common area, for further processing by you. You can identify which file each record came from using the FROM clause and a previously defined data name to hold the identity of that record. Additionally, a special IF statement (IF REPLICA) can test whether the record is a duplicate of the prior record or not.

Using a MERGE saves on the overhead that a SORT requires; the order in which the files are named in the MERGE statement determines the order of duplicate merged records. No work files or special memory requirements are necessary. The files to be merged should be in ascending key sequence, however.

Syntax for the MERGE Operation

All files referenced in the MERGE must be defined using the usual FILE and field definition statements in preceding statements. The format of the MERGE statement is:

```

MERGE  filename1 KEY1 dataname [KEY2 dataname KEY3 dataname...KEY9 dataname]
       filename2 KEY1 dataname [KEY2 dataname KEY3 dataname...KEY9 dataname]
       [filenameN KEY1 dataname {KEY2 dataname KEY3 dataname...KEY9 dataname}]
AREA   dataname [FROM dataname] [LENGTH dataname]
       [USING {DY282MER|subroutine}]

```

Special IF statements here

Figure 649 Format of the MERGE Statement

MERGE	Required. Declares the application as a MERGE operation.
filename1	Required. This file name is the name of the first file designated for the MERGE operation. The file must be defined in preceding statements.
KEY1 dataname	Required. The data name is used as the first key field for the MERGE operation. filename1 must be in ascending sequence by this key.
KEY2 dataname, KEY3 dataname through KEY9 dataname	Optional. These data names identify the other key fields to be used for the MERGE operation. If specified, filename1 must be in ascending key sequence by these keys.
filename2	Required. This file name is the name of the second file designated for the MERGE operation. The file must be defined in preceding statements.
filenamen	Optional. This file name can identify any number of additional files to be included in the MERGE operation.
KEY1 dataname	Required if a file name has been specified as filenamen. If specified, filenamen must be in ascending sequence by this key.
KEY2 dataname, KEY3 dataname through KEY9 dataname	Optional. These data names identify the other key fields for the file named in filenamen. If specified, filenamen must be in ascending key sequence by these keys.
AREA dataname	Required. Identifies the area in which the output records from the MERGE are placed. This data name can be any previously defined character field large enough to hold the merged records, that is, it must be at least as large as the largest record length of the files being merged.
FROM dataname	Optional. Identifies the file from which any given merged record was obtained. VISION:Results automatically places the name of the file from which the merged record was obtained in this field. This data name must be previously defined as an 8-byte character field.
LENGTH dataname	Optional. Identifies the length of the merged record. The VISION:Results data name automatically places the length of the merged record in this field. This data name must be a previously defined field, with a type of numeric, packed, or binary.
Using DY282MER	Optional. The default is DY282MER. Identifies the type of MERGE operation to use. You can also specify the name of your own subroutine to be used for the merging.

Recommendations and Requirements for Merging

Adhere to the following standards when merging:

- Specify at least two files during a MERGE.
- Each file in the MERGE must have at least one key specified.
- Every file in the MERGE must have the same number of keys specified.
- Each respective key in each file specified in the MERGE must be of the same type and length. For example, KEY1 on file1 and KEY1 on file2 must be of the same type and length and have the same number of decimal positions.
- Every file in the MERGE must be previously defined in a file definition.
- For all files in a MERGE, the record format and record length entries must be specified in their respective file definitions.
- For all files in a MERGE, it is recommended that the files be in ascending sequence by their respective keys.
- SORT is disallowed during a request with a MERGE. However, the files can be sorted in a prior request and then passed to the current request.
- MERGE is not allowed with OPTION STRUCTURED.
- An output file cannot be OUTPUT FROM a file that is referenced in a MERGE statement.

Merge Conditions in the Procedure Logic

VISION:Results automatically reads the files to be merged in a particular manner depending upon the keyword you select in the USING clause of the MERGE statement. The records are then presented to the procedure logic for further testing or manipulation by you. A special IF statement (IF REPLICA) is available for you to check on the condition of the MERGE during the automatic cycle. This special IF statement is explained in detail later in this section.

There are two factors that govern the manner in which the files are read. The first factor is the order in which the files to be merged are entered in the MERGE statement. The first file in the MERGE has the highest precedence, and so on. The second factor is the value of the keys at the beginning of each cycle.

The Rule for DY282MER

A record is read from every file once. The record with the lowest key value is chosen to be output as the first merged record. If the keys are the same, the record from the file with the highest precedence is chosen. Another record is then read from this file and the comparison for the lowest key is done again. The process continues until end of file on all files. Unlike the MATCH operation, this operation does no look aheads.

Example of Using DY282MER

Assume three files are being merged as follows:

```
FILE TRAN1 VB 160
  T1REC 160
  T1KEY1 10 20
FILE TRAN2 VB 72
  T2REC 72
  T2KEY1 10 1
FILE TRAN3 VB 88
  T3REC 88
  T3KEY1 10 5

FILE OUTFILE VB 160 LENGTH OUTLEN OUTPUT FROM OUTFILE
  OUTREC 160

WORKAREA
  SOURCEFL 8

MERGE  TRAN1 KEY1 T1KEY1
      TRAN2 KEY1 T2KEY1
      TRAN3 KEY1 T3KEY1
      AREA OUTREC FROM SOURCEFL LENGTH OUTLEN
```

Your procedure logic goes here ...

Figure 650 Example of Using DY282MER

The keys on the three files are as follows (duplicate keys on a file are shown here with an apostrophe '):

TRAN1	TRAN2	TRAN3
2	1	2
3	2	4
5	2'	5
8	4	8
9	8	9
		11

The records output to the procedure logic at each point in the cycle are as follows:

AREA	OUTREC	SOURCEFL
1		TRAN2
2		TRAN1
2		TRAN2
2'		TRAN2
2		TRAN3
3		TRAN1
4		TRAN2
4		TRAN3
5		TRAN1
5		TRAN3
8		TRAN1
8		TRAN2
8		TRAN3
9		TRAN1
9		TRAN3
11		TRAN3

The logic can be understood by walking through some of the steps. A record is read from all three files (2 1 2). Because TRAN2 has the next lowest key, its record is moved to the OUTREC area and another record is read from TRAN2 (2). Because the keys are all equal (2 2 2), the record moved is from the file with the highest precedence TRAN1 (2) and a record is read from TRAN1 (3). Because the lowest key value of (3 2 2) is 2 and TRAN2 and TRAN3 have the same key value, the next record is moved from TRAN2 (2), another record is read from TRAN2, and so on.

IF REPLICA

The special IF REPLICA statement in your procedure logic allows you to test whether the current merged record has the same key value as the previous record. This is similar to testing for duplicate keys on the merged file, except that the first key in a set of duplicate keys is not considered to be a duplicate. If the test is successful, the condition is true; otherwise, the condition is false.

The format of the IF REPLICA statement is:

```
IF [NOT] REPLICA [AND|OR condition . . .] imperative . . . ENDIF
```

Figure 651 Format of the IF REPLICA Statement

For example, the following IF statements are valid:

```
IF REPLICA REJECT ENDIF
```

```
IF NOT REPLICA AND TIKEY1 GT 5 ACCEPT ENDIF
```

The IF REPLICA statement can be used with any other conditional expressions using the AND and OR connectors in the usual way. This IF statement is like any other IF statement in VISION:Results. It can be negated by saying IF NOT REPLICA. The IF REPLICA statement can be used with any of the other special IF statements to test for combinations of conditions.

The following example illustrates the points in the MERGE cycle when the IF REPLICA statement is true. Suppose the following three files are used in the MERGE operation:

TRAN1	TRAN2	TRAN3
2	1	2
3	2	4
5	2'	5
8	4	8
9	8	9
		11

The following merged records input to the procedure logic are tested with an IF REPLICA statement for a true or false condition:

AREA	OUTREC	SOURCEFL	IF REPLICA
1		TRAN2	False
2		TRAN1	False
2		TRAN2	True
2'		TRAN2	True
2		TRAN3	True
3		TRAN1	False
4		TRAN2	False

4	TRAN3	True
5	TRAN1	False
5	TRAN3	True
8	TRAN1	False
8	TRAN2	True
8	TRAN3	True
9	TRAN1	False
9	TRAN3	True
11	TRAN3	False

The first key in a set of keys that are equal in value is not considered to be a replica. This is because the current record key value is compared to the previous record key value for equality.

Merge Example

Two transaction files, one sorted and the other unsorted, are merged and the resulting merged records listed.

The unsorted transaction file is sorted in the first request and passed to the second request using the PICNSAVE and USE statements.

The second request, REPORT2, merges the two sorted files and lists the contents of the merged file.

Merging a sorted and unsorted file:

```
FILE TRAN1 FB 80 STATUS T1STAT
  TRAN1REC 80
  TRAN1ID  9 1
  TRAN1NAM  8
  TRAN1SAL  5 NU

SORT TRAN1 USING TRAN1ID

PICNSAVE SAVE1 USING TRAN1ID TRAN1NAM TRAN1SAL BY 40

REPORT2

FILE TRAN2 FB 80 STATUS T2STAT
  TRAN2REC 80
  TRAN2ID  9 1
  TRAN2ED  8
  TRAN2EXP  5 NU

USE SAVE1 STATUS T1STAT

WORKAREA
  OUTREC 80
MERGE  SAVE1 KEY1 TRAN1ID
       TRAN2 KEY1 TRAN2ID
       AREA OUTREC
IF T1STAT EQ 'E' AND T2STAT EQ 'E' STOP ENDIF
LIST OUTREC
T1 'CONTENTS OF THE MERGED FILE'
```

Figure 652 Merge Example

Error Analysis During Match and Merge

Special error handling is provided for you during MATCH and MERGE. If you encounter an execution time error (0C7 – data exception) during your run and are interested in analyzing the record that caused the error during the match or merge, follow the steps outlined here.

1. Go to page 2 of your listing and look at the error message shown. If you spot the cause of the error right away, fix it and rerun the program. If you need more information about the cause of the error, go to Step 2.
2. Rerun the program using the following OPTION statement as the first statement in your program:

```
OPTION XREFA PRINTER
```

This statement produces a cross-reference listing and a mini-dump in addition to the standard file # key # field identification.

Go to page 2 of your listing and look at the error message. The example below shows the message produced when bad data is encountered in a file.

```
COMPLETION CODE-0299      *** ERROR OCCURRED DURING THIS RUN ***
```

```
MATCH/MERGE FILE 01, KEY 1 (CURR) HAS CAUSED AN ERROR.  
IT SHOULD BE A VALID PACKED FIELD BUT IT LOOKS LIKE THIS:
```

```
048508385  
FFFFFFFFF  
048508385
```

THE DATA MAY NOT BE DEFINED PROPERLY IN THE FIELD DEFINITION STATEMENT. CHECK THE DEFINITION AGAINST A FILE LAYOUT OR FILE PRINTOUT AND MAKE SURE THE FIELD IS THE CORRECT LENGTH, LOCATION AND DATA TYPE.

IF THE FIELD CONTAINS ALL HIGH-VALUES (FF), THE FILE MAY BE AT END OF FILE. CHECK THE PROGRAM STATEMENTS TO SEE IF DATA IS BEING ACCESSED IN THE INPUT AREA AFTER END OF FILE. ONE REASON MIGHT BE THAT YOU HAVE SPECIFIED IN A LIST STATEMENT TO PRINT A FIELD FROM THE INPUT AREA AT 'GRAND TOTAL TIME' (ON FINAL).

PROBABLE CAUSES:

- 1) DATA FIELD DOES NOT START IN THE POSITION INDICATED, OR THE AREA CODE IS INCORRECT.
- 2) SIZE OF FIELD IS INCORRECT.
- 3) DATA TYPE GIVEN IN PARAMETER DOES NOT CORRESPOND TO THAT CONTAINED IN THE RECORD. (CHECK PACKED FIELDS)
- 4) NUMERIC FIELD CONTAINS INVALID CHARACTERS.
- 5) DATA BEING PROCESSED FROM INPUT FILE THAT HAS REACHED EOF. RECORD NOW CONTAINS HIGH VALUES.

Figure 653 Error Analysis During Match and Merge

The message indicates that FILE 01, KEY1 of the current record has caused the error. The message could have indicated (NEXT) for the next record instead of (CURR) as having caused the error. Note this information about the file and the record.

3. Go to the cross-reference listing on the next page and find the location and area of the key that represents KEY1 of FILE 01. Remember that FILE 01 is the first file designated in the MATCH or MERGE statement.

In this example, FILE 01 is the file IMASTER and the current KEY1 field is IMKEY (the first key on the MATCH statement).

If the error message indicated (NEXT) instead of (CURR), you look in the cross-reference listing for DYLNFO1K1 as the data name. This is VISION:Results' way of naming the key for the NEXT record for file 1. Similarly, other NEXT keys might be named DYLNFO1K2 for the second key of the first file, and so on. The naming convention for the NEXT keys in the cross-reference listing is as follows:

DYLNFnKn

where:

Fnn is the number of the file in the MATCH or MERGE statement.

Kn is the number of the key in the MATCH or MERGE statement.

Note the area and location information. In this case, the location is 378 and the area is R.

4. Go to the listing of the dump and find the R area or REFORMAT area. Count off 378 bytes as the starting location of the record that caused the error. With the information you now have, you can analyze what fields in the record might have caused the error. In this case, it happened to be the first key field of a file that was defined as packed but did not have packed data.

Writing Your Own MATCH Subroutine

As explained earlier, VISION:Results provides you with the additional capability of writing your own MATCH and MERGE subroutines. There are three separate methods of file advancement provided with the MATCH and MERGE facility. See [Match Conditions in the Procedure Logic on page 643](#). These techniques should be sufficient to handle all your match needs. However, if they are not adequate, the information provided in this section instructs you about how to write your own routine to replace DY282MAT. As an alternative, you can contact Technical Support at <http://ca.com/support> to request new file advancement techniques.

DY282MAT Invocation

DY282MAT is invoked once for each special MATCH statement found in your VISION:Results program (such as IF MATCHED...), with the exception of the IF ADVANCED... flag, which is set depending on whether or not a read occurred for a given file. It is also called once at program startup (first-cycle call) and once at the beginning of each cycle (new-cycle call).

DY282MAT MATCH Subroutine

A series of parameters is passed to DY282MAT (see [Parms Passed to DY282MAT on page 672](#)); your MATCH subroutine receives these same parameters. You do not have to be concerned with I/O because it is not done within the MATCH subroutine itself. During the first-cycle, depending on the file advancement technique (for a further explanation of the file advancement techniques, see [Match Conditions in the Procedure Logic on page 643](#)), the read ahead (also known as next read) flags are set to get a next and current set of keys for the first new-cycle call.

Also, although it is not demonstrated in the sample program, the first-cycle call establishes the method of file advancement. The new-cycle call occurs following any file reads and prior to the MATCH keywords (for an overview of the MATCH logic flow, see the flowchart in [Generated from Your Special MATCH Logic Statements on page 667](#)). The new-cycle call sets the next read flags based on the file advancement technique specified. It also sets an ALL MATCHED flag on if all keys across all files match. It also updates an array that is used to keep track of duplicates by file.

DY282MAT is called each time one of the special MATCH statements (for example, IF MATCHED..., IF LASTDUP..., and so on) is referenced. Your custom subroutines must perform all the current functions of DY282MAT and attend to all of the details such as parameters, flags, and codes (see [Parms Passed to DY282MAT on page 672](#)). DY282MAT uses them to determine whether to test for matches, duplicates, and so on, but sets only the following:

- The read ahead flag on the first-/new-cycle calls
- The status code on your match keyword calls

The VISION:Results MATCH logic can be thought of as an extension of the automatic cycle.

The following section shows the flowchart and corresponding pseudo code that you can use as an example for your own subroutines.

For a flowchart that is an expanded detail illustration of code generated by the required MATCH... statement in your program, see [Generated from Your Special MATCH Logic Statements on page 667](#). For details of this MATCH statement, see [Syntax for the MATCH Operation on page 641](#).

This flowchart describes new-cycle logic (the new-cycle concept is described below), executed at the beginning of every cycle.

The first-time logic at the top of the flowchart is executed only once, at the beginning of program execution (the rest of the logic is executed once per automatic cycle). The additional USER CODE at the bottom of the flowchart is VISION:Results code generated by all other special MATCH and MERGE statements, such as IF MATCHED..., IF DUPLICATE..., and so on.

- FIRST-CYCLE** A first-cycle call is a special call consisting of ON ONE code that is executed only once (during the execution of a program) for the purpose of correctly positioning the files for the first new-cycle call. DY282MAT is called (with function code A) to set the read ahead flags. These are set depending on the file advancement technique used. For information about how files are advanced differently depending on this option, see [Match Conditions in the Procedure Logic on page 643](#). For the match logic flowchart, see [Generated from Your Special MATCH Logic Statements on page 667](#).
- NEW-CYCLE** A new-cycle call is a special call issued during the beginning of each cycle (following any FILE READs and prior to your match logic) that sets the next read flags for the next automatic cycle. These are set depending on the file advancement technique used. For information about how files are advanced differently depending on this option, see [Match Conditions in the Procedure Logic on page 643](#). It also sets an array of duplicate flags for each file used if the match keywords IF DUPLICATE, IF FIRSTDUP, or IF LASTDUP are specified. It also sets a flag that is used if you specify IF MATCHED. Our program demonstrating a sample USERMAT routine for SETREAD ORIGINAL does not perform these last two functions.

Generated from Your Special MATCH Logic Statements

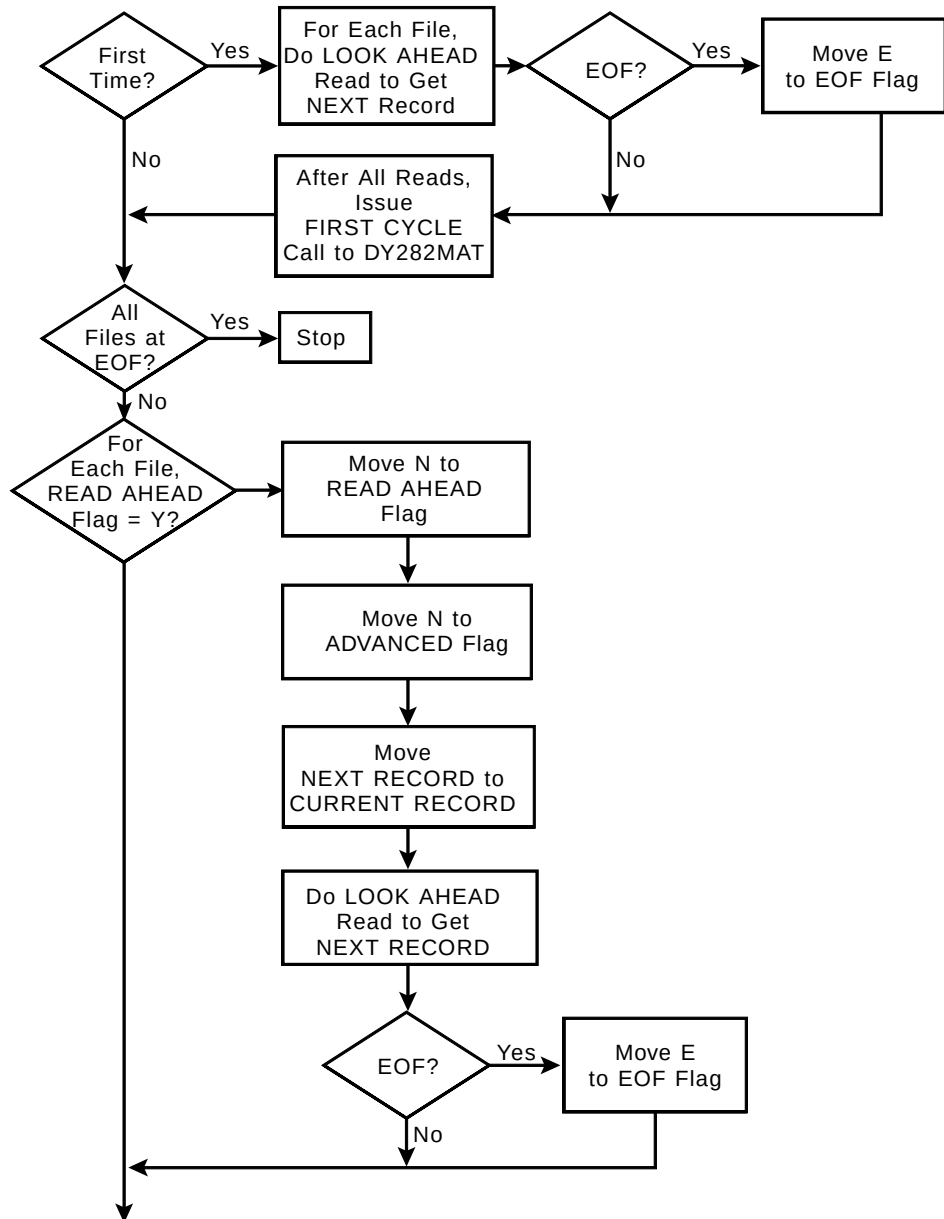
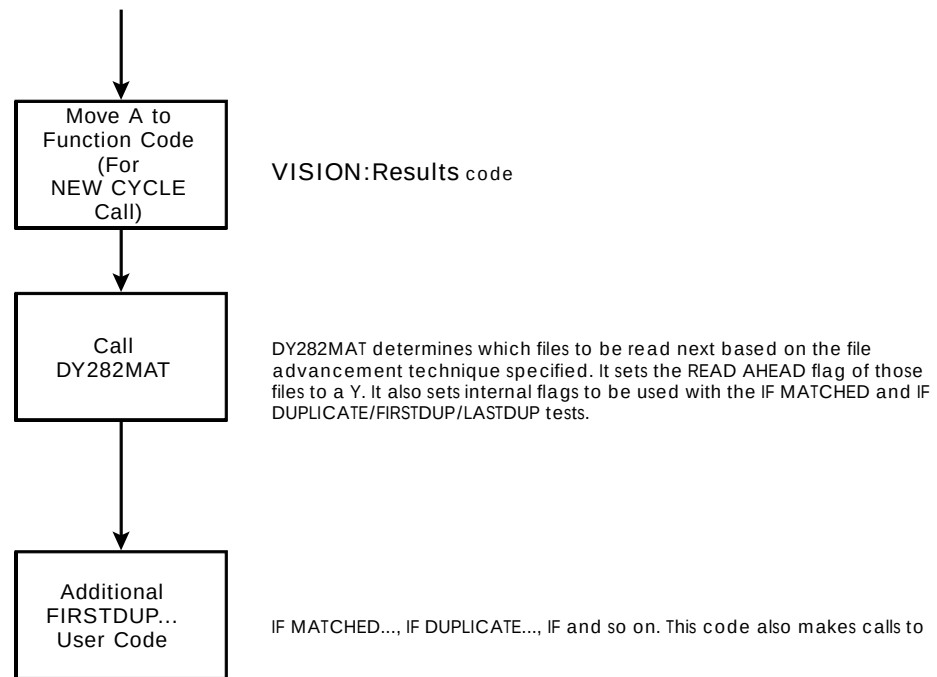


Figure 654 MATCH Logic Flow Diagram (Page 1 of 2)



MATCH Logic (calls generated from user code)

Figure 654 MATCH Logic Flow Diagram (Page 2 of 2)

All of these MATCH and MERGE statements (with the exception of IF ADVANCED...) generate a call to DY282MAT.

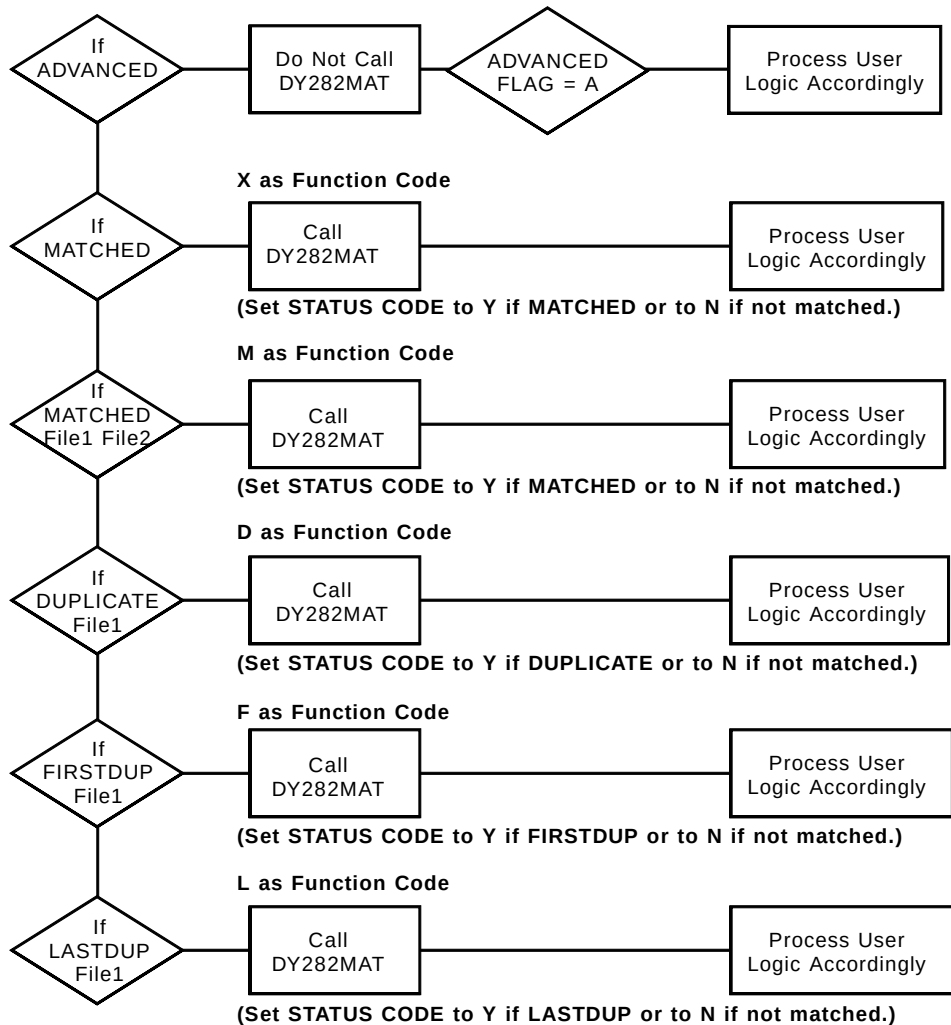


Figure 655 MATCH/MERGE Statements

In the first example, IF ADVANCED..., DY282MAT is NOT called. The ADVANCED FLAG is set to an A if a file has been advanced, or a ' ' (blank) if it has not been advanced.

```

IF ADVANCED filea
PRINT 'filea advanced'
ELSE
PRINT 'filea not read'
ENDIF
  
```

← If ADVANCED flag is set to an A, do this logic.
 Otherwise, do this logic.

The other MATCH statements generate calls to DY282MAT because they require the setting of the status code. Function codes are passed to DY282MAT. DY282MAT then performs an action corresponding to the function code and sets the status code to a Y or an N.

```
IF MATCHED      ← If status code is set to a Y, do this logic.
PRINT 'allfiles matched'
ELSE            Otherwise, do this logic.
PRINT 'allfiles not matched'
ENDIF
```

Examples for each statement are shown at the beginning of [Parms Passed to DY282MAT on page 672](#).

ON ONE Logic

This is first-cycle logic that is executed only once during the execution of a program, doing an initial read from each file so that you have next-record keys for each file before the actual processing takes place. DY282MAT is called to correctly position the current-record keys by setting the read ahead flags.

Initially, the read ahead flag is set as a result of the first-cycle call to DY282MAT. An E in the next read flag indicates that end of file has been reached.

VISION:Results Cycle For MATCH Run Before Your Statements

Perform these steps:

1. Reset each Advanced Flag to a blank.
2. If all files are at end of file, a STOP is issued.
3. If the Read Ahead flag is set to a Y, do the following:
 - Set the Read Ahead flag to an N.
 - Set the Advanced Flag to an A.
 - Move the next record to the current record.
 - Read the next record—if at end of file, set Read Ahead flag to E.
4. Issue a call to DY282MAT using function code A for new-cycle logic (for information about a new-cycle call, see [Writing Your Own MATCH Subroutine on page 664](#)). DY282MAT compares file keys and sets the read ahead flag of the next files to a Y for the next cycle through. This read does not take place until the next cycle.
5. To avoid the possibility of an endless loop, be sure your subroutine sets one or more of the Read Flags to Y for read ahead so that another record is always read in.
6. To avoid the possibility of an abend because of a read after end of file, be sure that your subroutine leaves all Read Flags set to E alone (this indicates that end of file is reached for those files).

User Statements with Other Special MATCH Logic

IF ADVANCED File1

A True or False flag that indicates that a new record has been read from the specified file on this cycle, checking the Advanced Flag for an A.

IF MATCHED

A True or False flag that indicates whether all keys across files are equal and tests for matching across all files by generating a call to DY282MAT using function code X. If all files match, the status code is set to a Y.

IF MATCHED File1 File2

A special True or False flag that indicates whether all keys across any two specified files are equal. It tests for matching across the files by generating a call to DY282MAT using function code M. If matching occurred between both files, the status code is set to a Y.

IF DUPLICATE File1

A True or False flag that indicates whether the current key is equal to either the previous key, the next key, or both. It tests for duplicates on a given file by generating a call to DY282MAT using function code D. If the current keys are equal to those of either the previous keys, next keys, or both, the status code is set to a Y.

IF FIRSTDUP File1

A True or False flag that is set to true if the current key is equal to the next key but NOT the previous key. This command tests for first duplicate on two or more keys in a given file by generating a call to DY282MAT using function code F. If the current keys are equal to the next keys and not equal to the previous keys, the status code is set to a Y.

IF LASTDUP File1

A True or False flag that is set to true if the current key is equal to the previous key but NOT the next key. This command tests for last duplicate on two or more keys in a given file by generating a call to DY282MAT using function code L. If the current keys are equal to the previous keys and not equal to the next keys, the status code is set to a Y.

These command statements are described in more detail in [Match Conditions in the Procedure Logic on page 643](#). For a complete description of all parameters, including the flags, function codes, and status codes for DY282MAT, see [Parms Passed to DY282MAT on page 672](#).

Parms Passed to DY282MAT

PARM1

1st byte	Function Code (A, X, M, D, F, or L). A= Generates a new-cycle call to DY282MAT. X= Checks for a match across ALL files (sets Status Code). M= Checks for a match between two files (sets Status Code). D= Checks for duplicates on a given file (sets Status Code). F= Checks for first duplicate on two or more keys on a given file (current and next keys) (sets Status Code). L= Checks for last duplicate on two or more keys on a given file (current and previous keys) (sets Status Code).
2nd byte	Status Code (set to Y or N by DY282MAT according to one of the above actions).
3rd byte	Number of files being matched (in binary).
4th byte	Number of keys per file (in binary).
5th byte	File number to be checked (in binary). Used with: IF DUPLICATE File 1 IF MATCHED File 1 File 2 IF FIRSTDUP File 1 IF LASTDUP File 1 (where a binary 1 represents File 1)
6th byte	File number to be checked (in binary). Used with: IF MATCHED File 1 File 2 (where a binary 2 represents File 2)
7th byte	Read Ahead flag for File 1 (set by DY282MAT, initially Y). Y= Read ahead WILL take place on next cycle. N= Read ahead will NOT take place on next cycle. E= End of file has been reached.
8th byte	Advanced flag for File 1. A= Advance to another record has taken place on current cycle. BLANK= Advance to another record has NOT taken place. Used with: IF ADVANCED for File 1
9th byte	Read Ahead flag for File 2 (set by DY282MAT; the same rules as 7th byte).

10th byte Advanced flag for File 2 (the same rules as 8th byte).

.
.
.

and so on.

All of these parameters are passed to DY282MAT. The status code and next read flags can be modified by DY282MAT and passed back to the program.

PARM2

The second parm passed consists of two or more 3-byte fields. Each field contains the following key information:

1st byte Type of key:

- C Character.
- B Binary numeric.
- N Zoned-decimal numeric.
- P Packed numeric.

2nd byte One byte actual length of the key in binary.

3rd byte Number of decimals (zoned-decimal); for character fields, leave this field blank.

There is one 3-byte field for each file key. For example, assuming two files (each with two keys), there would be 12 bytes of storage (four keys) at Parm2. The layout is as follows:

File 1 – Key 1

File 1 – Key 2

File 2 – Key 1

File 2 – Key 2

.
.
.

and so on.

PARM3 through PARM10

With two files and two keys per file (four keys), the layout of keys being passed to the subroutine (starting with Parm3) is as follows:

PARM 3:	File 1	Current	Key 1
PARM 4:	File 1	Current	Key 2
PARM 5:	File 1	Next	Key 1

PARM 6:	File 1	Next	Key 2
PARM 7:	File 2	Current	Key 1
PARM 8:	File 2	Current	Key 2
PARM 9:	File 2	Next	Key 1
PARM 10:	File 2	Next	Key 2

The following listing of USERMAT is one example of a replacement subroutine for the built-in match routine, DY282MAT. It works exactly as DY282MAT does (including the rules of matching), with one exception: it processes only two input files, each with two keys.

USERMAT is designed as a starting point from which you can write your own specialized match processing subroutine according to rules different from those of DY282MAT. The output using this subroutine is identical to that generated using DY282MAT. The sample program demonstrates the SETREAD ORIGINAL file advancement technique (see [Match Conditions in the Procedure Logic on page 643](#)). It is your responsibility to add other file advancement techniques as needed.

```

31
***** USERMAT STARTS HERE
*
* THIS IS A SIMPLIFIED EXAMPLE OF A SUBROUTINE TO REPLACE
* 'DY282MAT' AS YOU WOULD WRITE IT.
*
* REGISTER CONVENTION :
*   R3  -> 1ST PARM      - FUNCTION CODE, STATUS
*                               CODE...
*   R4  -> 2ND PARM      - 12 BYTES OF KEY
*                               INFORMATION
*   R0,R1,R2,R5 THRU R14,R15 - WORK REGISTERS
*   R10 -> PARM ADDR
*   R11 -> BASE REGISTER 2
*   R12 -> BASE REGISTER 1
*
R0      EQU 0
R1      EQU 1
R2      EQU 2
R3      EQU 3
R4      EQU 4
R5      EQU 5
R6      EQU 6
R7      EQU 7
R8      EQU 8
R9      EQU 9
R10     EQU 10
R11     EQU 11
R12     EQU 12
R13     EQU 13
R14     EQU 14
R15     EQU 15
EJECT
USERMAT CSECT
        STM R14,R12,12(R13)      SAVE REGS 14 THRU 12
        LR  R12,R15              R12 AS BASE 1
        USING USERMAT,R12,R11
        LA  R11,4095(R12)        R11 AS BASE 2
        LA  R11,1(R11)
        ST  R13,SAVE+4           SAVE HSA/REG13
        LR  R15,R13
        LA  R13,SAVE              OUR 13 NOW SAVE
        ST  R13,8(R15)           SAVE LSA
        LR  R10,R1               SAVE REG 1
        LM  R3,R4,0(R10)         LOAD PARM REGS

```

Figure 656 USERMAT Example (Page 1 of 5)

```

        USING PARM1,R3          MAP REG 3 TO PARM 1
        USING PARM2,R4          MAP REG 4 TO PARM 2
        SPACE 1
        MVI STATUS,C'N'        INITIALIZE STATUS FLAG
        CLI FUNCTION,C'A'      COMPARE KEYS FOR READ AHEAD?
        BE CKNEXTRD            YES.
        CLI FUNCTION,C'X'      IF MATCHED? (E.G. ALL MATCHED)
        BE CKALLMAT            YES.
        CLI FUNCTION,C'M'      IF MATCHED FILE1 FILE2?
        BE CKANYMAT            YES.
        CLI FUNCTION,C'D'      IF DUPLICATE FILEX?
        BE CKDUPL              YES.
        CLI FUNCTION,C'F'      IF FIRSTDUP FILEX?
        BE CK1STDUP            YES.
        CLI FUNCTION,C'L'      IF LASTDUP FILEX?
        BE CKLSTDUP            YES.
* DOES NOT COME HERE
        DC H'0'                ABEND WITH '0C1'.
        EJECT
*
* FUNCTION CODE 'A' :
* - ALWAYS 1ST CALL TO 'DY282MAT' (OR USER SUBROUTINE)
* - SETS UP AREAS FOR PREVIOUS,CURRENT, AND NEXT KEYS FOR
*   EACH FILE
* - CONTROLS NEXT READ OF A FILE BY MOVING 'Y' TO
*   CORRESPONDING READ AHEAD FLAG (AND
*   PREVENTS AN ENDLESS LOOP)
* * NOTE: DO NOT RESET THE READ FLAG IF IT'S ALREADY
*   'E'.
*
* NOTE: 'DY282MAT' HANDLES A VARIABLE NUMBER OF KEYS ON A
* VARIABLE NUMBER OF FILES. THIS ROUTINE KNOWS IT
* HAS 2 FILES EACH WITH 2 KEYS.
*
MOVEIT MVC 0(0,R8),0(R9)      VARIABLE LENGTH MOVE.
CKNEXTRD DS 0H
        CLI ADVANCE1,C'A'      FILE 1 ADVANCED THIS CYCLE?
        BNE CKNEXTR2          NO.
        MVC FILE1PRV,FILE1CUR  YES...SAVE PREVIOUS KEY OF FILE1
CKNEXTR2 DS 0H
        CLI ADVANCE2,C'A'      FILE 2 ADVANCED THIS CYCLE?
        BNE CKNEXTR3          NO.
        MVC FILE2PRV,FILE2CUR  YES...SAVE PREVIOUS KEY OF FILE2
CKNEXTR3 DS 0H
        SR R7,R7              CLEAR REGISTER 7
* MOVE FILE1 CURRENT KEY1
        ICM R7,B'0001',KY1SIZE1 R7 = KEY LNG
        BCTR R7,R0            SUBTRACT 1 TO GET MACHINE LNG
        LA R8,FILE1CUR        R8 -> CURRENT FILE1 KEY
        L R9,8(R10)           R9 -> 3RD PARM
        EX R7,MOVEIT          MOVE FILE1 CURRENT KEY1
* MOVE FILE1 CURRENT KEY2
        AR R8,R7              R8 -> NEW CURRENT FILE1 KEY LOC
        LA R8,1(R8)
        ICM R7,B'0001',KY1SIZE2 R7 = KEY LNG
        BCTR R7,R0            SUBTRACT 1 TO GET MACHINE LNG
        L R9,12(R10)          R9 -> 4TH PARM
        EX R7,MOVEIT          MOVE FILE1 CURRENT KEY2
        SR R7,R7              CLEAR REGISTER 7
* MOVE FILE1 NEXT KEY1
        ICM R7,B'0001',KY1SIZE1 R7 = KEY LNG
        BCTR R7,R0            SUBTRACT 1 TO GET MACHINE LNG
        LA R8,FILE1NXT        R8 -> NEXT FILE1 KEY
        L R9,16(R10)          R9 -> 5TH PARM
        EX R7,MOVEIT          MOVE FILE1 NEXT KEY1
* MOVE FILE1 NEXT KEY2
        AR R8,R7              R8 -> NEW NEXT FILE1 KEY LOC
        LA R8,1(R8)
        ICM R7,B'0001',KY1SIZE2 R7 = KEY LNG
        BCTR R7,R0            SUBTRACT 1 TO GET MACHINE LNG
        L R9,20(R10)          R9 -> 6TH PARM
        EX R7,MOVEIT          MOVE FILE1 NEXT KEY2
        SR R7,R7              CLEAR REGISTER 7
* MOVE FILE2 CURRENT KEY1
        ICM R7,B'0001',KY2SIZE1 R7 = KEY LNG
        BCTR R7,R0            SUBTRACT 1 TO GET MACHINE LNG

```

Figure 656 USERMAT Example (Page 2 of 5)

```

        LA      R8,FILE2CUR      R8 -> CURRENT FILE2 KEY
        L       R9,24(R10)       R9 -> 7TH PARM
        EX      R7,MOVEIT       MOVE FILE2 CURRENT KEY1
* MOVE FILE2 CURRENT KEY2
        AR      R8,R7           R8 -> NEW CURRENT FILE2 KEY LOC
        LA      R8,1(R8)
        ICM     R7,B'0001',KY2SIZE2  R7 = KEY LNG
        BCTR    R7,R0           SUBTRACT 1 TO GET MACHINE LNG
        L       R9,28(R10)       R9 -> 8TH PARM
        EX      R7,MOVEIT       MOVE FILE2 CURRENT KEY2
        SR      R7,R7           CLEAR REGISTER 7
* MOVE FILE2 NEXT KEY1
        ICM     R7,B'0001',KY2SIZE1  R7 = KEY LNG
        BCTR    R7,R0           SUBTRACT 1 TO GET MACHINE LNG
        LA      R8,FILE2NXT      R8 -> NEXT FILE2 KEY
        L       R9,32(R10)       R9 -> 9TH PARM
        EX      R7,MOVEIT       MOVE FILE2 NEXT KEY1
* MOVE FILE2 NEXT KEY2
        AR      R8,R7           R8 -> NEW NEXT FILE2 KEY LOC
        LA      R8,1(R8)
        ICM     R7,B'0001',KY2SIZE2  R7 = KEY LNG
        BCTR    R7,R0           SUBTRACT 1 TO GET MACHINE LNG
        L       R9,36(R10)       R9 -> 6TH PARM
        EX      R7,MOVEIT       MOVE FILE2 NEXT KEY2
        CLI     RDAHEAD1,C'E'     FILE 1 AT EOF?
        BE      MOVE2RD          YES...SET READ AHEAD FILE2.
        CLI     RDAHEAD2,C'E'     FILE 2 AT EOF?
        BE      MOVE1RD          YES...SET READ AHEAD FILE1.
        CLC     FILE1CUR,FILE2CUR  COMPARE FILE 1/FILE 2 CURR KEYS
        BL      MOVE1RD          FILE1 LESS? SET READ AHEAD FILE1
        BH      MOVE2RD          FILE2 LESS? SET READ AHEAD FILE2
        CLC     FILE1NXT,FILE2NXT  COMPARE FILE 1/FILE 2 NEXT KEYS
        BH      MOVE2RD          FILE2 LESS? SET READ AHEAD FILE2
        B       MOVE1RD          FILE1 LESS? (OR DEFAULT)
MOVE1RD DS      0H
        CLI     RDAHEAD1,C'E'     EOF REACHED?
        BE      PGMEXIT          YES...LEAVE IT ALONE.
        MVI     RDAHEAD1,C'Y'     SET FILE1 READ AHEAD TO 'Y'
        B       PGMEXIT
MOVE2RD DS      0H
        CLI     RDAHEAD2,C'E'     EOF REACHED?
        BE      PGMEXIT          YES...LEAVE IT ALONE.
        MVI     RDAHEAD2,C'Y'     SET FILE2 READ AHEAD TO 'Y'
        B       PGMEXIT
        EJECT

*
* FUNCTION CODE 'X' :
*   - 'IF MATCHED' SPECIFIED.
*   - SETS STATUS CODE TO 'Y' IF ALL FILES MATCH.
*
*
* NOTE: 'DY282MAT' HANDLES A VARIABLE NUMBER OF FILES. AS
* IT'S BUILDING THE KEYS, IT SETS A FLAG IF 'ALL
* MATCHED'.
* THIS ROUTINE IS JUST A SIMPLE CHECK BETWEEN THE
* CURRENT KEYS OF EACH FILE.
*
*
        CKALLMAT DS      0H
        CLC     FILE1CUR,FILE2CUR  FILES EQUAL?
        BE      PGMEXIT2          YES..SET STATUS CODE TO 'Y'
        B       PGMEXIT          NO..JUST EXIT.
        EJECT

*
* FUNCTION CODE 'M' :
*   - 'IF MATCHED FILE1 FILE2' SPECIFIED.
*   - SETS STATUS CODE TO 'Y' IF BOTH FILES MATCH.
*
*
* NOTE: 'DY282MAT' HANDLES ANY 2 FILES SPECIFIED. YOU CAN
* PICK UP WHICH 2 FILES ARE BEING COMPARED FROM
* PARM1 BYTES 5 AND 6.
* THIS ROUTINE IS JUST A SIMPLE CHECK BETWEEN THE
* CURRENT KEYS OF EACH FILE.
*
        CKANYMAT DS      0H

```

Figure 656 USERMAT Example (Page 3 of 5)

```

        CLC    FILE1CUR,FILE2CUR      FILES EQUAL?
        BE     PGMEXIT2              YES...SET STATUS CODE TO 'Y'
        B      PGMEXIT              NO....JUST EXIT.
        EJECT

*
* FUNCTION CODE 'D' :
*   - 'IF DUPLICATE FILEX' SPECIFIED.
*   - SETS STATUS CODE TO 'Y' IF DUPLICATE ON FILE
*     SPECIFIED.
*
* NOTE: 'DY282MAT' HANDLES CHECKING ON ANY FILE SPECIFIED.
*       THIS ROUTINE KNOWS IT'S EITHER FILE1 OR FILE2.
*
CKDUPL  DS     0H
        TM     MATCHFL1,X'01'        CK ON FILE1?
        BO     CKDUPL1              YES.
        B      CKDUPL2              NO.
CKDUPL1 DS     0H
        CLC    FILE1CUR,FILE1PRV      CURRENT = PREVIOUS?
        BE     PGMEXIT2              YES...SET STATUS CODE TO 'Y'
        CLC    FILE1CUR,FILE1NXT      CURRENT = NEXT?
        BE     PGMEXIT2              YES...SET STATUS CODE TO 'Y'
        B      PGMEXIT              NO....JUST EXIT.
CKDUPL2 DS     0H
        CLC    FILE2CUR,FILE2PRV      CURRENT = PREVIOUS?
        BE     PGMEXIT2              YES...SET STATUS CODE TO 'Y'
        CLC    FILE2CUR,FILE2NXT      CURRENT = NEXT?
        BE     PGMEXIT2              YES...SET STATUS CODE TO 'Y'
        B      PGMEXIT              NO....JUST EXIT.
        EJECT

*
* FUNCTION CODE 'F' :
*   - 'IF FIRSTDUP FILEX' SPECIFIED.
*   - SETS STATUS CODE TO 'Y' IF FIRST DUPLICATE ON FILE
*     SPECIFIED.
*
* NOTE: 'DY282MAT' HANDLES CHECKING ON ANY FILE SPECIFIED.
*       THIS ROUTINE KNOWS IT'S EITHER FILE1 OR FILE2.
*
CK1STDUP DS    0H
        TM     MATCHFL1,X'01'        CK ON FILE1?
        BO     CK1STDU1              YES.
        B      CK1STDU2              NO.
CK1STDU1 DS    0H
        CLC    FILE1CUR,FILE1PRV      CURRENT = PREVIOUS?
        BE     PGMEXIT              YES...CAN'T BE FIRST DUPE
        CLC    FILE1CUR,FILE1NXT      CURRENT = NEXT?
        BE     PGMEXIT2              YES...SET STATUS CODE TO 'Y'
        B      PGMEXIT              NO....JUST EXIT.
CK1STDU2 DS    0H
        CLC    FILE2CUR,FILE2PRV      CURRENT = PREVIOUS?
        BE     PGMEXIT              YES...CAN'T BE FIRST DUPE
        CLC    FILE2CUR,FILE2NXT      CURRENT = NEXT?
        BE     PGMEXIT2              YES...SET STATUS CODE TO 'Y'
        B      PGMEXIT              NO....JUST EXIT.
        EJECT

*
* FUNCTION CODE 'L' :
*   - 'IF LASTDUP FILEX' SPECIFIED.
*   - SETS STATUS CODE TO 'Y' IF LAST DUPLICATE ON FILE
*     SPECIFIED.
*
* NOTE: 'DY282MAT' HANDLES CHECKING ON ANY FILE SPECIFIED.
*       THIS ROUTINE KNOWS IT'S EITHER FILE1 OR FILE2.
*
CKLSTDUP DS    0H
        TM     MATCHFL1,X'01'        CK ON FILE1?
        BO     CKLSTDU1              YES.
        B      CKLSTDU2              NO.
CKLSTDU1 DS    0H
        CLC    FILE1CUR,FILE1NXT      CURRENT = NEXT?
        BE     PGMEXIT              YES...CAN'T BE LAST DUPE
        CLC    FILE1CUR,FILE1PRV      CURRENT = PREVIOUS?
        BE     PGMEXIT2              YES...SET STATUS CODE TO 'Y'
        B      PGMEXIT              NO....JUST EXIT.
CKLSTDU2 DS    0H

```

Figure 656 USERMAT Example (Page 4 of 5)

```

        CLC     FILE2CUR,FILE2NXT      CURRENT = NEXT?
        BE      PGMEXIT                YES...CAN'T BE LAST DUPE
        CLC     FILE2CUR,FILE2PRV      CURRENT = PREVIOUS?
        BE      PGMEXIT2               YES...SET STATUS CODE TO 'Y'
        B        PGMEXIT                NO....JUST EXIT.
        EJECT

*
* RETURN WITH RETURN CODE '0'
*
PGMEXIT2 DS    0H
        MVI     STATUS,C'Y'           SET STATUS FLAG TO 'Y'
PGMEXIT  DS    0H
        L       R13,SAVE+4            RESTORE HSA/REG13
        LM      R14,R12,12(R13)       RESTORE R14 THRU R12
        LA      R15,0                 SET RETURN CODE TO 0
        BR      R14                   RETURN
        SPACE 3

* DATA AREAS
SAVE     DS    18F                    SAVE AREA
* NOTE: ALTHOUGH WE ARE LIMITING TOTAL SIZE OF ANY
*       COMBINATION OF KEYS TO 256 BYTES,
*       NO SUCH LIMIT IN 'DY282MAT'|
FILE1PRV DC    XL256'00'              FILE1 PREVIOUS KEYS
FILE1CUR DC    XL256'00'              FILE1 CURRENT KEYS
FILE1NXT DC    XL256'00'              FILE1 NEXT KEYS
FILE2PRV DC    XL256'00'              FILE2 PREVIOUS KEYS
FILE2CUR DC    XL256'00'              FILE2 CURRENT KEYS
FILE2NXT DC    XL256'00'              FILE2 NEXT KEYS
        EJECT

* DSECTS
PARM1    DSECT
FUNCTION DS    CL1                    FUNCTION CODE (A,X,M,D,F,L)
STATUS   DS    CL1                    STATUS CODE (Y OR N)
FILES    DS    XL1                    NO OF FILES BEING MATCHED
KEYS      DS    XL1                    NO OF KEYS PER FILE
MATCHFL1 DS    XL1                    FILE 1 MATCH NUMBER
MATCHFL2 DS    XL1                    FILE 2 MATCH NUMBER
* NOTE: FROM THIS PT FORWARD WOULD BE VARIABLE NUMBER OF
*       FLAGS DEPENDING ON NUMBER OF FILES.
RDAHEAD1 DS    CL1                    FILE 1 READ AHEAD FLAG
ADVANCE1 DS    CL1                    FILE 1 ADVANCE FLAG
RDAHEAD2 DS    CL1                    FILE 2 READ AHEAD FLAG
ADVANCE2 DS    CL1                    FILE 2 ADVANCE FLAG
        SPACE 3
PARM2    DSECT
* NOTE: FROM THIS PT FORWARD WOULD BE VARIABLE NUMBER OF KEY
*       ATTRIBUTES DEPENDING ON NUMBER OF FILES.
KY1ATTR1 DS    0CL3                  FILE1 KEY1 -
ATTRIBUTES
KY1TYPE1 DS    CL1                    - TYPE (C,B,N,P)
KY1SIZE1 DS    XL1                    - ACTUAL LENGTH
KY1DEC1  DS    CL1                    - NO OF DECIMALS
KY1ATTR2 DS    0CL3                  FILE1 KEY2 -
ATTRIBUTES
KY1TYPE2 DS    CL1                    - TYPE (C,B,N,P)
KY1SIZE2 DS    XL1                    - ACTUAL LENGTH
KY1DEC2  DS    CL1                    - NO OF DECIMALS
KY2ATTR1 DS    0CL3                  FILE2 KEY1 - ATTRIBUTES
KY2TYPE1 DS    CL1                    - TYPE (C,B,N,P)
KY2SIZE1 DS    XL1                    - ACTUAL LENGTH
KY2DEC1  DS    CL1                    - NO OF DECIMALS
KY2ATTR2 DS    0CL3                  FILE2 KEY2 -
ATTRIBUTES
KY2TYPE2 DS    CL1                    - TYPE (C,B,N,P)
KY2SIZE2 DS    XL1                    - ACTUAL LENGTH
KY2DEC2  DS    CL1                    - NO OF DECIMALS
USERMAT  CSECT
        LTORG
        END     USERMAT
***** USERMAT ENDS HERE

```

Figure 656 USERMAT Example (Page 5 of 5)

Writing Your Own MERGE Subroutine

This section shows you how to merge according to rules different from the ones set by DY282MER. You must be familiar with IBM host assembler, assembler programming conventions, and the VISION:Results MATCH and MERGE feature.

The MERGE logic provided with VISION:Results should be satisfactory for most of your requirements. It is able to handle multiple keys on multiple files automatically. The MERGE logic is simpler than the MATCH logic because it does not do any look aheads. You must provide a work area into which the input files are merged. The MERGE logic then determines which key is lowest and selects the record from that file next. You have control over the following items:

- Whether or not that record is written using access to the data area itself (for example, AREA dataname).
- The file name from which the merge takes place (for example, FROM dataname).
- The length of the data merged (for example, LENGTH dataname).
- Whether or not it is an exact duplicate of the previous record (for example, IF REPLICA...).

Your custom MERGE subroutine must be capable of handling all necessary MERGE logic. The reasons that you may require your own custom MERGE logic are the same as those for MATCH subroutines. The rules for MERGE file Advancement can be found in [Merge Conditions in the Procedure Logic on page 659](#).

The following section describes the process by which DY282MER merges files. The parameters, flags, and function codes are explained in detail so you know exactly what your MERGE subroutines must do to successfully replace DY282MER.

MERGE Subroutine

As with the MATCH subroutine, the MERGE subroutine is passed a series of parameters (your MERGE subroutines require the same parameters). You do not have to be concerned with I/O, as it is not done within the MERGE subroutine itself. The first time through, the MERGE subroutine is passed all the keys of all the files and updates one Read Flag per file, that indicates which file to read next. This is the same type of new-cycle call made by the MATCH subroutine.

The subroutine is called again each time a MERGE statement is referenced (IF REPLICA is the only statement currently available). Your custom subroutines must perform all the current functions of DY282MER and attend to all of the details such as parameters, flags, and codes. The next section shows a flowchart and corresponding pseudo code from which you can create your own subroutines.

MERGE Logic (Automatic Cycle)

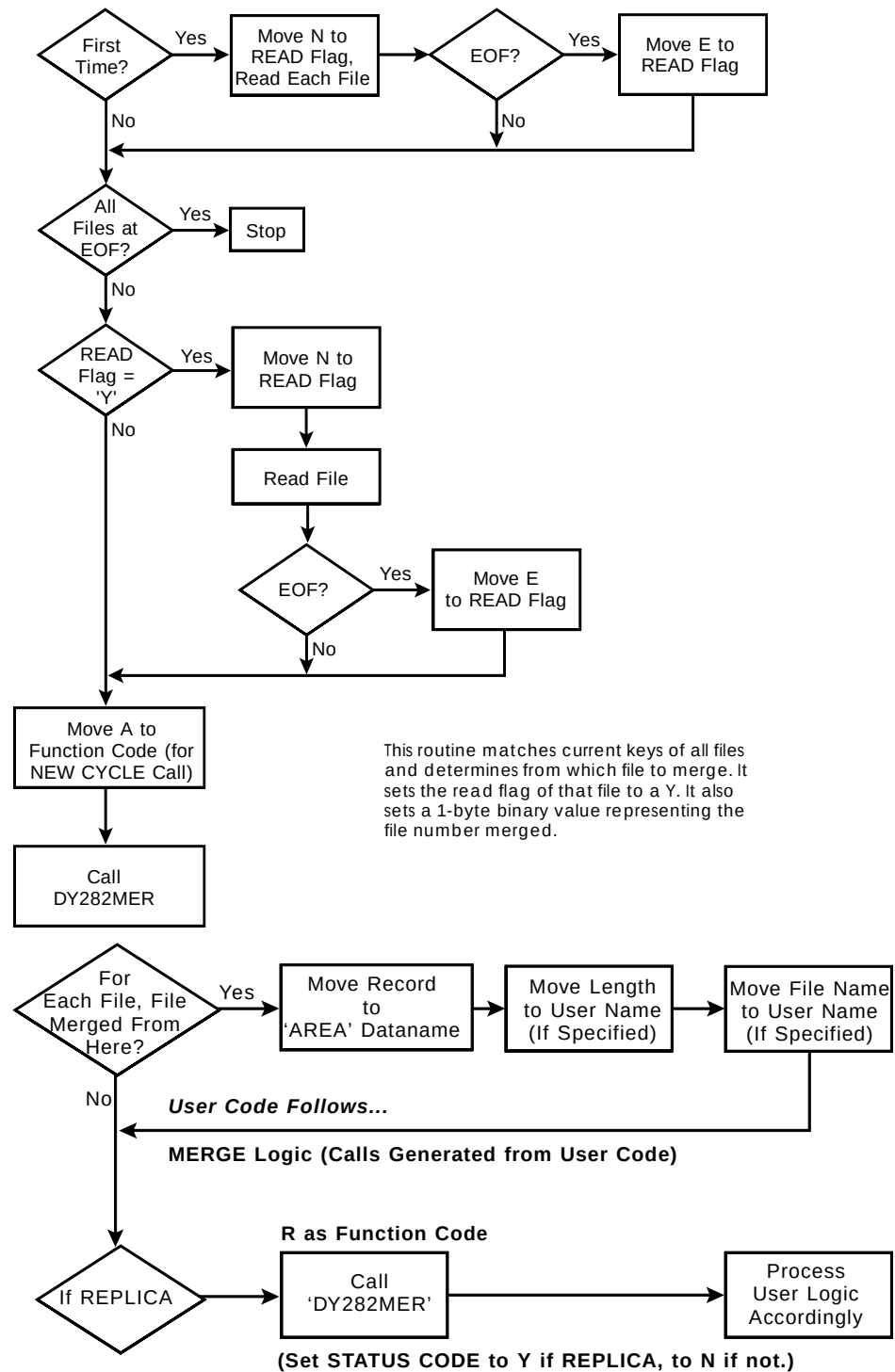


Figure 657 MERGE Logic Flow Diagram

IF REPLICA is the only MERGE statement currently available. Logic using this statement requires the setting of the status code. A function code of (R), shown in the diagram above, is passed to DY282MER, which then tests for replication of record keys and sets the status code to a Y or an N.

```
IF REPLICA  ← If the STATUS CODE is Y, do this logic.
    PRINT 'identical to last'
ELSE       ← Otherwise, do this logic.
    PRINT 'not identical'
ENDIF
```

ON ONE Logic

Set read flag.

Cycle for MERGE Run Before Your Statements

Use these steps:

1. If all files are at end of file, a STOP is issued.
2. If the Read Flag is set to a Y, do the following:
 - Reset the Read Flag to an N.
 - Read a record (when end of file is reached, set the Read Flag to an E).
3. Issue a new-cycle call to DY282MER using function code A. A new-cycle call is explained in [DY282MAT Invocation on page 664](#). DY282MER compares file keys and returns a binary value indicating the file number from which to merge. An X'01' in the file number field indicates a merge from the first file, an X'02' indicates a merge from the second file, and so on.
Be careful writing your own subroutines. One of the following items **MUST** be done to avoid creating an endless loop or abending after end of file:
 - The subroutine must set one or more of the Read Flags to a Y for a read.
 - The subroutine must leave all Read Flags set to E alone (this indicates that end of file is reached for those files).
4. When a file number is returned, move the appropriate data as follows:
 - Move the input record to the specified output area.
 - Move the input length to the specified output length name (optional).
 - Move the input file name (eight characters) to the specified output file name (optional).

User Statements with Other Special MERGE Logic

IF REPLICA

Tests for duplicates on a given file by generating a call to DY282MER using function code R. If the current keys are equal to the previous keys on the records being merged, the status code is set to a Y.

Parms Passed to DY282MER

PARM1

1st byte	Function Code (A or R).
A=	Generates a new-cycle call to DY282MER.
R=	Checks for IF REPLICA; if found, cycle repeats without falling through to your code.
2nd byte	Status Code (set to Y or N by DY282MER).
3rd byte	Number of files being merged (in binary).
4th byte	Number of keys per file (in binary).
5th byte	Returned 1-byte binary value representing the file number merged (where a binary 1 indicates File 1).
6th byte	N/A.
7th byte	Read flag for File 1 (set by DY282MER).
Y=	Another record is read-in.
N=	Another record is NOT to be read-in.
E=	End of file has been reached.
8th byte	N/A.
9th byte	Read flag for File 2 (the same rule as 7th byte).
10th byte	N/A.
.	
.	
.	
	and so on.

PARM2

As with the earlier example, the second parm consists of one or more 3-byte fields. See [Parms Passed to DY282MAT on page 672](#). Each field contains the following key information:

1st byte Type of key: C Character.
 B Binary numeric.
 N Zoned decimal.
 P Packed decimal.

2nd byte One byte actual length of the key in binary.

3rd byte Number of decimals (zoned decimal); for character fields, leave this byte blank.

There is one 3-byte field for each file-key. For example, assuming two files (each with two keys), there are 12 bytes of storage (four keys) at Parm2. The layout is as follows:

File 1 – Key 1
 File 1 – Key 2
 File 2 – Key 1
 File 2 – Key 2
 and so on.

All of these parameters are passed to DY282MER. The read flag, status code, and 1-byte binary value (byte 5) can be modified by DY282MER and passed back to the VISION:Results program.

Each of the four 3-byte fields belonging to the keys listed above correlates to a key being passed to the subroutine, as shown below.

PARM3 through PARM6:

PARM3: File 1 Current Key 1
 PARM4: File 1 Current Key 2
 PARM5: File 2 Current Key 1
 PARM6: File 2 Current Key 2

Although the number of parms used is only half the number used in the MATCH example, this layout still represents four keys and the 12 corresponding bytes of information because MERGE does not use read ahead logic.

USERMER is an example of a MERGE subroutine designed to replace DY282MER. Like USERMAT, it conducts MERGE processing on only two files, each with two keys. As with USERMAT, USERMER is the starting point from which you can write your own specialized MERGE processing subroutine. The output using this subroutine is identical to that generated using DY282MER.

```
***** USERMER STARTS HERE
*
* REGISTER CONVENTION :
* R3  ->  1ST PARM           - FUNCTION CODE, STATUS
*                                CODE...
* R4  ->  2ND PARM           - 12 BYTES OF KEY
*                                INFORMATION
* R0,R1,R2,R5 THRU R14,R15 - WORK REGISTERS
* R10 ->  PARM ADDR
* R11 ->  BASE REGISTER 2
* R12 ->  BASE REGISTER 1
*
R0      EQU  0
R1      EQU  1
R2      EQU  2
R3      EQU  3
R4      EQU  4
R5      EQU  5
R6      EQU  6
R7      EQU  7
R8      EQU  8
R9      EQU  9
R10     EQU 10
R11     EQU 11
R12     EQU 12
R13     EQU 13
R14     EQU 14
R15     EQU 15
EJECT
USERMER CSECT
        STM   R14,R12,12(R13)      SAVE REGS 14 THRU 12
        LR    R12,R15              R12 AS BASE 1
        USING USERMER,R12,R11
        LA    R11,4095(R12)        R11 AS BASE 2
        LA    R11,1(R11)
        ST    R13,SAVE+4           SAVE HSA/REG13
        LR    R15,R13
        LA    R13,SAVE             OUR 13 NOW SAVE
        ST    R13,8(R15)           SAVE LSA
        LR    R10,R1              SAVE REG 1
        LM    R3,R4,0(R10)         LOAD PARM REGS
        USING PARM1,R3             MAP REG 3 TO PARM 1
        USING PARM2,R4             MAP REG 4 TO PARM 2
        SPACE 1
        MVI   STATUS,C'N'          INITIALIZE STATUS FLAG
        CLI   FUNCTION,C'A'        COMPARE KEYS FOR NEXT READ?
        BE    CKNEXTRD             YES.
        CLI   FUNCTION,C'R'        IF REPLICA?
        BE    CKREPL               YES.
* DOES NOT COME HERE
        DC    H'0'                 ABEND WITH '0C1'.
        EJECT
*
* FUNCTION CODE 'A' :
* - ALWAYS 1ST CALL TO 'DY282MER' (OR your SUBROUTINE)
* - SETS UP AREAS FOR CURRENT KEYS FOR EACH FILE
* - CONTROLS NEXT READ OF A FILE BY MOVING 'Y' TO
*   CORRESPONDING READ FLAG (AND PREVENTS AN
*   ENDLESS LOOP)
* * NOTE: DO NOT RESET THE READ FLAG IF IT'S ALREADY
*   'E'.
* - MOVES A X'01' OR X'02' TO FLAG WHICH INDICATES FILE TO
*   MERGE FROM (E.G. X'01' = FILE1, X'02' = FILE2 ...).
* - SETS A FLAG INDICATING WHETHER OR NOT CURRENT MERGED
*   RECORD IS AN EXACT REPLICA OF PREVIOUS MERGED RECORD
* - MOVES CURRENT MERGED RECORD KEYS TO PREVIOUS
*
* NOTE: 'DY282MER' HANDLES A VARIABLE NUMBER OF KEYS ON A
```

Figure 658 USERMER Example (Page 1 of 3)

```

*      VARIABLE NUMBER OF FILES. THIS ROUTINE KNOWS IT
*      HAS 2 FILES EACH WITH 2 KEYS.
*
MOVEIT  MVC    0(0,R8),0(R9)
CKNEXTRD DS    0H
        MVI     REPLFLAG,C'N'      INITIALIZE 'IF REPLICA' FLAG
        MVI     MERGEFLX,X'FF'     INITIALIZE FILE X MERGE NUMBER
        SR      R7,R7              CLEAR REGISTER 7
* MOVE FILE1 CURRENT KEY1
        ICM     R7,B'0001',KY1SIZE1 R7 = KEY LNG
        BCTR    R7,R0              SUBTRACT 1 TO GET MACHINE LNG
        LA      R8,FILE1CUR        R8 -> CURRENT FILE1 KEY
        L       R9,8(R10)          R9 -> 3RD PARM
        EX      R7,MOVEIT          MOVE FILE1 CURRENT KEY1
* MOVE FILE1 CURRENT KEY2
        AR      R8,R7              R8 -> NEW CURRENT FILE1 KEY LOC
        LA      R8,1(R8)
        ICM     R7,B'0001',KY1SIZE2 R7 = KEY LNG
        BCTR    R7,R0              SUBTRACT 1 TO GET MACHINE LNG
        L       R9,12(R10)         R9 -> 4TH PARM
        EX      R7,MOVEIT          MOVE FILE1 CURRENT KEY2
        SR      R7,R7              CLEAR REGISTER 7
* MOVE FILE2 CURRENT KEY1
        ICM     R7,B'0001',KY2SIZE1 R7 = KEY LNG
        BCTR    R7,R0              SUBTRACT 1 TO GET MACHINE LNG
        LA      R8,FILE2CUR        R8 -> CURRENT FILE2 KEY
        L       R9,16(R10)         R9 -> 5TH PARM
        EX      R7,MOVEIT          MOVE FILE2 CURRENT KEY1
* MOVE FILE2 CURRENT KEY2
        AR      R8,R7              R8 -> NEW CURRENT FILE2 KEY LOC
        LA      R8,1(R8)
        ICM     R7,B'0001',KY2SIZE2 R7 = KEY LNG
        BCTR    R7,R0              SUBTRACT 1 TO GET MACHINE LNG
        L       R9,20(R10)         R9 -> 6TH PARM
        EX      R7,MOVEIT          MOVE FILE2 CURRENT KEY2
        CLI     RDNEXT1,C'E'       FILE 1 AT EOF?
        BE      MOVE2RD           YES...SET READ NEXT FILE2.
        CLI     RDNEXT2,C'E'       FILE 2 AT EOF?
        BE      MOVE1RD           YES...SET READ NEXT FILE1.
        CLC     FILE1CUR,FILE2CUR  COMPARE FILE 1/FILE 2 CURR KEYS
        BNH     MOVE1RD           FILE1 LESS OR EQUAL?
        B       MOVE2RD           SET FILE1
                                   FILE2 LESS?
                                   SET FILE2
MOVE1RD DS    0H
        CLI     RDNEXT1,C'E'       EOF REACHED?
        BE      PGMEXIT           YES...LEAVE IT ALONE.
        MVI     RDNEXT1,C'Y'       SET FILE1 READ NEXT TO 'Y'
        MVI     MERGEPLX,X'01'     SET TO MERGE FROM FILE 1
        CLC     FILEXPRV,FILE1CUR  CURRENT KEY = SAVED?
        BNE     MOVE1RDX          NO.
        MVI     REPLFLAG,C'Y'     YES...SET FOR 'IF REPLICA'.
MOVE1RDX MVC    FILEXPRV,FILE1CUR  MOVE CURRENT TO SAVED.
        B       PGMEXIT
MOVE2RD DS    0H
        CLI     RDNEXT2,C'E'       EOF REACHED?
        BE      PGMEXIT           YES...LEAVE IT ALONE.
        MVI     RDNEXT2,C'Y'       SET FILE2 READ NEXT TO 'Y'
        MVI     MERGEFLX,X'02'     SET TO MERGE FROM FILE 2
        CLC     FILEXPRV,FILE2CUR  CURRENT KEY = SAVED?
        BNE     MOVE2RDX          NO.
        MVI     REPLFLAG,C'Y'     YES...SET FOR 'IF REPLICA'.
MOVE2RDX MVC    FILEXPRV,FILE2CUR  MOVE CURRENT TO SAVED.
        B       PGMEXIT
        EJECT
*
* FUNCTION CODE 'R' :
*   - 'IF REPLICA' SPECIFIED.
*   - SETS STATUS CODE TO 'Y' IF CURRENT MERGED RECORD KEY
*     WAS AN EXACT DUPLICATE OF PREVIOUS MERGED RECORD KEY.
*
CKREPL  DS    0H
        CLI     REPLFLAG,C'Y'     REPLICA?
        BE      PGMEXIT2          YES...SET STATUS CODE TO 'Y'
        B       PGMEXIT          NO....JUST EXIT.
        EJECT

```

Figure 658 USERMER Example (Page 2 of 3)

```

*
* RETURN WITH RETURN CODE '0'
*
PGMEXIT2 DS    0H
          MVI    STATUS,C'Y'          SET STATUS FLAG TO 'Y'
PGMEXIT   DS    0H
          L      R13,SAVE+4           RESTORE HSA/REG13
          LM     R14,R12,12(R13)      RESTORE R14 THRU R12
          LA     R15,0                 SET RETURN CODE TO 0
          BR     R14                  RETURN
          SPACE 3
* DATA AREAS
SAVE      DS    18F                  SAVE AREA
REPLFLAG  DC    C'N'                 REPLICA FLAG
* NOTE: ALTHOUGH WE ARE LIMITING TOTAL SIZE OF ANY
*       COMBINATION OF KEYS TO 256 BYTES, NO
*       SUCH LIMIT IN 'DY282MER'|
FILEXPRV  DC    XL256'00'            FILEX PREVIOUS KEYS
FILE1CUR  DC    XL256'00'            FILE1 CURRENT KEYS
FILE2CUR  DC    XL256'00'            FILE2 CURRENT KEYS
          EJECT
* DSECTS
PARM1     DSECT
FUNCTION  DS    CL1                  FUNCTION CODE (A,X,M,D,F,L)
STATUS   DS    CL1                  STATUS CODE (Y OR N)
FILES    DS    XL1                  NO OF FILES BEING MATCHED
KEYS     DS    XL1                  NO OF KEYS PER FILE
MERGEFLX DS    XL1                  FILE X MERGE NUMBER
FILLER1  DS    XL1
* NOTE: FROM THIS PT FORWARD WOULD BE VARIABLE NUMBER OF
*       FLAGS DEPENDING ON NUMBER OF FILES.
RDNEXT1  DS    CL1                  FILE 1 READ NEXT FLAG
FILLER2  DS    CL1
RDNEXT2  DS    CL1                  FILE 2 READ NEXT FLAG
FILLER3  DS    CL1
          SPACE 3
PARM2     DSECT
* NOTE: FROM THIS PT FORWARD WOULD BE VARIABLE NUMBER OF KEY
*       ATTRIBUTES DEPENDING ON NUMBER OF FILES.
KY1ATTR1 DS    0CL3                 FILE1 KEY1 - ATTRIBUTES
KY1TYPE1 DS    CL1                  - TYPE (C,B,N,P)
KY1SIZE1 DS    XL1                  - ACTUAL LENGTH
KY1DEC1  DS    CL1                  - NO OF DECIMALS
*
KY1ATTR2 DS    0CL3                 FILE1 KEY2 - ATTRIBUTES
KY1TYPE2 DS    CL1                  - TYPE (C,B,N,P)
KY1SIZE2 DS    XL1                  - ACTUAL LENGTH
KY1DEC2  DS    CL1                  - NO OF DECIMALS
*
KY2ATTR1 DS    0CL3                 FILE2 KEY1 - ATTRIBUTES
KY2TYPE1 DS    CL1                  - TYPE (C,B,N,P)
KY2SIZE1 DS    XL1                  - ACTUAL LENGTH
KY2DEC1  DS    CL1                  - NO OF DECIMALS
*
KY2ATTR2 DS    0CL3                 FILE2 KEY2 - ATTRIBUTES
KY2TYPE2 DS    CL1                  - TYPE (C,B,N,P)
KY2SIZE2 DS    XL1                  - ACTUAL LENGTH
KY2DEC2  DS    CL1                  - NO OF DECIMALS
*
USERMER   CSECT
          LTORG
          END    USERMER
***** USERMER ENDS HERE

```

Figure 658 USERMER Example (Page 3 of 3)

Chapter 25: Using Interval Selection

VISION:Results features an interval selection method capable of selecting every *n*th record from any number of files and/or work areas. This is useful in generating test data or in the creation of subfiles for testing purposes.

Defining Interval Selection

Use interval selection when you want to select every *n*th record from a given file's population. Rather than scan the file's population for specific records, you can select every *n*th record after a certain interval has been determined and a random starting point has been chosen.

Your first step is to determine the interval size wanted. This is done by dividing the population size by the sample size wanted.

Interval wanted =
$$\frac{\text{Population size (exact or estimated)}}{\text{Sample size wanted}}$$

The result interval wanted may or may not have a remainder. For example, if your selection criteria is select 1,000 records from a population of 15,000, the interval is 15.

Interval wanted =
$$\frac{15,000}{1,000}$$

You select every fifteenth record from your population. This in no way implies that selection should commence with the first record and then the sixteenth, thirty-first, and so on. Select a starting point, which in this case is a random number from 1 to 15, inclusive. After selecting your random starting number, either through some random method or random number tables, select every fifteenth record thereafter.

Assuming that our random starting number for the above example was 12, the first five selected records are the twelfth, twenty-seventh, forty-second, fifty-seventh, and seventy-second record in the file. VISION:Results automatically selects the *n*th record.

This discussion is based on a whole number being the result of the division. In the following example, the result of the division contains a remainder:

$$15.6 = \frac{15,600}{1,000}$$

In this case, the interval is 15.6. Always round down. If the resulting interval is 213.2, round down to 213 and select a random starting number from 1 to 213, inclusive. If the resulting interval wanted is 15.8, round down to 15 and select a random starting number from 1 to 15, inclusive

If you do not already have the following four file assignments, be sure to include them in the JCL of all programs that use the interval selection facility.

z/OS	VSE
AUDPRINT	SYS009
AUFCBF	SYS008
AUDEPF	SYS010
AUDWORK	SYS011

Internal Selection Criteria

Your VISION:Results program FILE statements and field definition statements must precede the statement defining your interval selection request. For further information on these topics, see the *Advantage VISION:Results for z/OS Getting Started* and [Defining Interval Selection on page 687](#).

Following are the formats to use when specifying the positional parameters for interval selection.

```
SAMPLE nn INTERVAL interval size desired  starting number  
      [maximum sample size]
```

Figure 659 Formats When Specifying the Positional Parameters for Interval Selection

SAMPLE

Required. Describes the application as selection. No abbreviations are allowed.

nn

Required. A unique application number must be assigned to each request. This number must be an integer value from 1 to 99.

INTERVAL

Required. Describes the function as interval selection. No abbreviations are allowed.

INTERVAL Size Desired

Required. The number you must supply to tell VISION:Results which n th record you want selected, for example, every tenth record or every twentieth record, and so on. This number must be an integer value from 1 to 999,999,999; the value cannot be zero. Commas are acceptable. The value cannot exceed nine digits.

Starting Number

Required. The number with which selection is to commence, selected randomly as described in [Defining Interval Selection on page 687](#). A numeric value from 1 to 999,999,999. Commas are acceptable. The value cannot exceed nine digits.

Maximum Sample Size

Optional. The total number of records you want to select from the file. Commas are acceptable; the value cannot be zero. If omitted, this value defaults to 999,999,999.

To invoke interval selection as defined in this chapter, specify the IF SAMPLING command, described in [IF SAMPLING Command on page 689](#), in your program procedure logic. If the invoking command is omitted, selection is not performed.

IF SAMPLING Command

The IF SAMPLING command, followed by the application number, invokes interval selection in your procedure logic. The unique application number ties this command to the interval selection statement and uses the criteria specified in that statement. For example:

```
VISION:Results FILE statement and  
field definition statements  
.  
SAMPLE 01 INTERVAL 10 5 100  
.  
IF SAMPLING 01 NEXT ELSE REJECT ENDIF  
.  
.
```

Figure 660 IF SAMPLING Command

If the IF SAMPLING statement is omitted, no selection is performed, even though the criteria was defined in a SAMPLE statement.

In the preceding example, every tenth record on the file (beginning with the fifth record), is selected, with a specified maximum sample size of 100. The IF SAMPLING command tells VISION:Results to accept the record if it meets the criteria specified in the selection statement. All other records are rejected.

IF SAMPLING Command Examples

Example 1 Randomly Select Every Fiftieth Record

Select every fiftieth record until 500 records have been selected or end of file is reached, whichever comes first. Selection is to commence with the tenth record and each selected record (every fiftieth record) from the input file is to be written to an output tape.

The following shows the VISION:Results statements necessary to generate your interval selection. The numbers at the left are for reference purposes only.

```
1  FILE ACCTREC FB 352 5280
2  FILE OUTREC OUTPUT FROM ACCTREC FB 352 5280

3  SAMPLE 01 INTERVAL 50 10 500

4  IF SAMPLING 01 ACCEPT ELSE REJECT ENDIF
```

Figure 661 Statements Necessary to Generate an Interval

Statements 1 and 2 are the FILE statements defining the input and output files. The FILE statement is described in [Chapter 6: Using the FILE Command](#). The program specifies that each record selected is written to an output file. The keyword OUTPUT FROM ACCTREC in the FILE OUTREC statement tells VISION:Results to use the ACCTREC file buffer (that contains the selected record) to create the output file.

In Statement 3, interval selection is specified, with an interval size of 50. Selection begins with the tenth record; thereafter, every fiftieth record is selected until the sample size equals 500 records or until end of file is reached.

The procedure logic, Statement 4, invokes interval selection 01, defined in Statement 3. Both the SAMPLE definition statement and IF SAMPLING command must be present in your program for the selection to be performed.

VISION:Results reads and bypasses the first nine records. Beginning with the tenth record, it selects every fiftieth record thereafter and accepts those records (writes each to the output file). If the record read is not one of the first nine records read or any fiftieth record thereafter, VISION:Results rejects it and reads another record.

A VISION:Results Non-Stratified Sampling Summary Report follows. This listing is automatically generated for every valid selection statement submitted to VISION:Results. The output contains all of the information submitted in the interval selection statement: application number, interval wanted, starting number, and maximum sample size (if specified). Finally, the summary lists the actual population size (total number of records on the file) and actual size of the sample selected.

```
1/15/99  VISION:RESULTS NON-STRATIFIED SAMPLING SUMMARY REPORT  PAGE 1

APPLICATION 1 - SIMPLE INTERVAL SAMPLING
INTERVAL DESIRED                                50
STARTING NUMBER                                  10
ACTUAL POPULATION SIZE                           200
MAXIMUM SAMPLE SIZE                              500
ACTUAL SAMPLE SIZE                                4
```

Figure 662 [Example 1](#) Output—Randomly Select Every Fiftieth Record

In this particular example, the actual population size was smaller than expected. This is apparent from the actual sample size selected from the input file. The following is a hexadecimal/graphic dump of the four records selected.

```

                                DATE 1/15/99      PAGE    2
4          5          6          7          8          9
012345678901234567890123456789012345678901234567890123456789

2677082677112277301097800000  9999970200  HILL,GARY E  10
FFFFFFFFFFFFFFFFFFFFFFFF44FFFFFFFF39070CCDD6CCCE4C44444
2677082677112277201097800000099999702005405C8933B71980500000
2877062877010278Z01227800000  9999970200  SIAS, EDWARD  60
FFFFFFFFFFFFFFFFFFFFFFFF44FFFFFFFFF07031ECCE64CCECD444
2877062877012278901277800000099999702000932C2912B05461946000
0977090977100677 00000000000 0001080200  %YEAGER,CLIFFORD 110
FFFFFFFFFFFFFFFF44FFFFFFFFF78946ECCCD6CDCCDDC4
097709097710067700000000000000010802000083C851759B339666940
1077031077092977 00000000000 9999970102  MYERS, EARL  160
FFFFFFFFFFFFFFFF44FFFFFFFFF72019DECDE64CCDD44444
1077031077092977000000000000099999701020121C48502B0519300000
```

Figure 663 Hexadecimal/Graphic Dump of Four Records

The numbers at the right on this dump show that record numbers 10, 60, 110, and 160 were selected. Based on the criteria of problem 1, every fiftieth record is selected after the initial starting number, 10, is chosen.

Example 2 Interval Selection with Three Random Starts

You have just completed a new program and want to test it on your personnel file. You decide to use the interval selection feature to create a smaller, more manageable version of the file, and then test your program against that subset of records. You decide to have three random starts, 3, 8, and 15, in order to minimize the possible bias inherent in interval selection. The size of the interval is 20 records. To accomplish this, three samples are selected simultaneously.

The following statements accomplish the task:

```
1 COPY DYAUDREC
2 FILE OUTFILE OUTPUT FROM ARFILE FB 352 5280

3 SAMPLE 01 INTERVAL 20 3
4 SAMPLE 02 INTERVAL 20 8
5 SAMPLE 03 INTERVAL 20 15

6 IF SAMPLING 01 WRITE OUTFILE ENDIF
7 IF SAMPLING 02 WRITE OUTFILE ENDIF
8 IF SAMPLING 03 WRITE OUTFILE
9 ELSE REJECT ENDIF
```

Figure 664 [Example 2](#) Interval Selection with Three Random Starts

The numbers at the left are for reference purposes only.

Statements 3, 4, and 5 specify the criteria for your three selection applications. All have a wanted interval size of 20. The starting numbers specified are 3, 8, and 15, respectively. That is, the first application starts with the third record on the file and selects every twentieth record after that, the second starts with the eighth record and takes every twentieth after that, and the third starts with the fifteenth record and takes every twentieth record thereafter. None of the applications has designated a maximum sample size, so records are selected until the ARFILE reaches end of file or until the default 999,999,999 sample size is reached.

The first IF statement (Statement 6) invokes the selection criteria defined in SAMPLE 01. It specifies that if the record being read meets the criteria of Statement 3, write that record to the output file; otherwise, fall through to IF SAMPLING 02. Statement 7 states that if the current record meets the criteria of the second selection application (Statement 4), write that record to the output file; otherwise, fall through to Statement 8. Finally, the third IF statement accepts all records that match the criteria of the third selection application, Statement 5, and writes them to the output file. All other records are rejected.

Your selection program generated the following Non-Stratified Sampling Summary Report. For each interval application, VISION:Results lists the interval specified, starting number, actual population size (total number of records on the file), maximum sample size (in this case, the default amount, because none was specified), and actual sample size.

```
1/15/99  VISION:RESULTS NON-STRATIFIED SAMPLING SUMMARY REPORT  PAGE 1

APPLICATION 1 - SIMPLE INTERVAL SAMPLING
INTERVAL DESIRED                                20
STARTING NUMBER                                  3
ACTUAL POPULATION SIZE                           200
MAXIMUM SAMPLE SIZE                             999,999,999
ACTUAL SAMPLE SIZE                               10

APPLICATION 2 - SIMPLE INTERVAL SAMPLING
INTERVAL DESIRED                                20
STARTING NUMBER                                  8
ACTUAL POPULATION SIZE                           200
```

Figure 665 [Example 2](#) Output—Interval Selection with Three Random Starts (Page 1 of 2)

MAXIMUM SAMPLE SIZE	999,999,999
ACTUAL SAMPLE SIZE	10
APPLICATION 3 - SIMPLE INTERVAL SAMPLING	
INTERVAL DESIRED	20
STARTING NUMBER	15
ACTUAL POPULATION SIZE	200
MAXIMUM SAMPLE SIZE	999,999,999
ACTUAL SAMPLE SIZE	10

Figure 665 Example 2 Output—Interval Selection with Three Random Starts
(Page 2 of 2)

Chapter 26: Using Random Selection

Use random selection when records in a file are to be selected completely at random from the total number of records on the file (the population). This provides assurance that each record in the population has an equal chance of being selected.

Input to VISION:Results

There are two formats of random selection: Known Universe and Unknown Universe.

Your VISION:Results program FILE and field definition statements must precede the statement defining your random selection request. For more information on these subjects, see *Advantage VISION:Results for z/OS Getting Started* and [Chapter 6: Using the FILE Command](#).

If you do not already have the following four file assignments, include them in the JCL of all programs that use the random selection facility.

z/OS	VSE
AUDPRINT	SYS009
AUFCBF	SYS008
AUDEPF	SYS010
AUDWORK	SYS011

For a description of the VSE PARM statement to override the SYS numbers and device types, see the *Advantage VISION:Results for VSE Utilities Guide*.

Known Universe

When using the known universe format, you either know or can accurately estimate the total number of records in the file (the population) and you want a predetermined sample size.

```
SAMPLE nn RANDOM population size sample size  
[SEED original seed number]
```

Figure 666 Random Selection with Universe Known

SAMPLE

Required. Describes the application as selection. Abbreviations are not allowed.

nn

Required. A unique application number must be assigned to each sample request. This number must be an integer value from 1 to 99.

RANDOM

Required. Describes the function as random selection. Abbreviations are not allowed.

Population Size

Required. Indicates the exact or estimated population (file) size. This number must be an integer value from 1 to 999,999,999. Commas are acceptable; decimal points are not permitted. The value cannot be zero.

The size of the sample is solely dependent upon an accurately estimated population size. If you do not have a fairly good idea of the actual population size, you might need to make a separate VISION:Results run against your files to obtain this information, or use Unknown Universe. For more information about using an unknown universe, see [Unknown Universe on page 697](#).

You can estimate the population size, but be careful in doing so. If your estimate is greater than the actual population size, you can get a sample that is smaller than the calculated sample size. Conversely, if your estimate is less than the actual population size, you can get a sample that is larger than the calculated sample size.

Sample Size

Required. Indicates the sample size to be randomly selected from the population. An integer value from 1 to 999,999,999; cannot be larger than population size. The value cannot be zero.

SEED Original Seed Number

Optional. If you decide to use a seed number, the keyword SEED preceding the number is required. The seed number is used in initializing the algorithm used in the random number generator.

By specifying the seed number from a previous run, the sequence of random numbers generated is always identical; consequently, an interesting run can be repeated. Therefore, specify the original seed number here if you want to exactly duplicate a prior run. You can specify a seed number of your choice, but a seed of zero is the same as not specifying a seed.

The maximum value is 9,999,999. When the seed number is omitted, a random seed number is generated by consulting the computer time clock. The random seed number is printed in the Sampling Summary Report.

To invoke random selection as defined in this section, you must specify the IF SAMPLING command in your program procedure logic. If you omit this command, selection is not performed.

Unknown Universe

When using the unknown universe format, you have no immediate way of predetermining the size of your file, but you know that the population is immense and you want a certain approximate percentage of that large population. When using this format, supply one field of information: the approximate percentage wanted of an infinite population or unknown universe.

```
SAMPLE nn  RANDOM  percent of population %  
      [SEED original seed number]
```

Figure 667 Random Selection with Universe Unknown

SAMPLE

Required. Describes the application as selection. Abbreviations are not allowed.

nn

Required. A unique application number must be assigned to each sample request. This number must be an integer value from 1 to 99.

RANDOM

Required. Describes the function as random selection. Abbreviations are not allowed.

Percent (%) of Population

Required. The approximate percentage of the infinite population or unknown universe. The percent sign is required. The decimal point is required when fractional percentages are used (for example, 95.5%). If a whole number percentage is wanted, do not use a decimal point in the leftmost position (for example, code 25%, not .25%). The minimum percentage allowed is .00001; the maximum allowed is 100%.

SEED Original Seed Number

Optional. However, if you decide to use a seed number, the keyword **SEED** preceding the number is required. The seed number is used to initialize the algorithm used in the random number generator.

By specifying the seed number from a previous run, the sequence of random numbers generated is always identical; consequently, an interesting run can be repeated. Specify the original seed number here if you want to exactly duplicate a prior run. You can specify a seed number of your choice. However, a seed of zero is the same as not specifying a seed.

The maximum value is 9,999,999. When the seed number is omitted, a random seed number is generated by consulting the computer time clock. The random seed number is printed in the Sampling Summary Report.

To invoke random selection as defined in this chapter, you must specify the **IF SAMPLING** command in your program procedure logic. If you omit this command, selection is not performed.

When selecting the percentage of the population wanted, beware of the term approximate. This implies that the sample size selected is approximately the percent requested from the population encountered. The larger the population, the closer the actual percent is to the requested percent. This is demonstrated in the table that follows, where test runs were made that encountered populations of 200, 1000, and 10,000.

Percent Desired:	5.1	26.5	1.29	59.1	98.33333	100.00
Number Randomly Selected from 200	9 4.50%	4 27.00%	3 1.50%	107 53.50%	198 99.00%	200 100%
Number Randomly Selected from 1000	52 5.20%	225 22.50%	13 1.30%	579 57.90%	976 97.60%	1000 100%
Number Randomly Selected from 10,000	507 5.07%	2676 26.76%	147 1.47%	5912 59.12%	9830 98.30%	10000 100%

Figure 668 A Test Run That Encountered Populations of 200, 1000, and 10,000

IF SAMPLING Command

The IF SAMPLING command, followed by the application number, invokes random selection in your procedure logic. The unique application number ties this command to your selection statement and uses the criteria you specify in that statement. For example:

```
FILE statement and
field definition statements
.
.
SAMPLE 01 RANDOM 2%
.
IF SAMPLING 01 ACCEPT ENDIF
```

Figure 669 IF SAMPLING Command

If you omit the IF SAMPLING command, no selection is performed, even though it has been defined in a SAMPLE statement.

In this example, you are randomly selecting 2% of the records in the file. The IF SAMPLING command tells VISION:Results to accept the record if it meets the criteria specified in the selection statement. All other records are rejected.

The population and sample size can be dynamically determined during the execution of the program, providing it is done BEFORE the first execution of the corresponding IF SAMPLING... command.

Random Selection Examples

Example 1 Randomly Select Records and Write to Tape

Refrigerated Polar Unit Corporation has a payroll file that contains 50,000 records (this is the population, either exact or estimated). Your task is to randomly select 450 records from this file and write these randomly selected records to an output tape.

The coded statements necessary to accomplish this task are shown below. Line numbers are for reference purposes only.

```
1  FILE ACCTREC FB 320 960
2  FILE OUTREC FB 320 960 OUTPUT FROM ACCTREC
3  SAMPLE 01 RANDOM 50,000 450
4  IF SAMPLING 01 ACCEPT ELSE REJECT ENDIF
```

Figure 670 [Example 1](#) Randomly Select Records and Write to Tape

The random selection definition statement, Statement 3, contains both the estimated population size for the known universe and the sample size.

Two FILE statements, Statements 1 and 2, are required to accomplish this task. One FILE statement defines the input file; the other FILE statement defines the output file (identified by the keyword OUTPUT following the file name). You specified that you wanted to use the ACCTREC file buffer (that contains the selected record) to create the output file, OUTREC.

Statement 4, the program procedure logic, invokes random selection 01 as defined in Statement 3. Both the SAMPLE definition statement and IF SAMPLING command must be present in your program for the selection to be performed. Utilizing VISION:Results' automatic functions for reading and end of file processing, you need only one procedure statement to code the selection logic.

When a record is randomly selected (true condition), ACCEPT the record (write the selected record to the output file). If the record is not randomly selected, REJECT the record (false condition) and ignore it. The reading of the input file is automatic and continues until the file is exhausted.

The program in this example, in effect, tells VISION:Results to select 450 records from the population of 50,000 using the random number selection facility. Then, if the record read is selected, accept it and write the selected record to the output tape. If the record is not selected, reject it and read another.

[Figure 671](#) is the VISION:Results Non-Stratified Sampling Summary Report. This listing is generated for every valid selection statement submitted to VISION:Results. The output shown below contains all of the information submitted on the selection statement.

The application number is application1 for random selection. The values submitted for population size (50,000) and sample size (450) are listed. Finally, the report lists the actual values for these parameters and the original seed number. This is the number to use (in your selection statement in a subsequent run) to duplicate this run.

```
1/15/99  VISION:RESULTS NON-STRATIFIED SAMPLING SUMMARY REPORT  PAGE 1
APPLICATION 1 - SIMPLE RANDOM SAMPLING
ESTIMATED POPULATION SIZE                50,000
ACTUAL POPULATION SIZE                   50,000
SAMPLE SIZE DESIRED                      450
ACTUAL SAMPLE SIZE                       450
ORIGINAL SEED NUMBER                     11,337
```

Figure 671 [Example 1](#) Output—Randomly Select Records and Write to Tape

VISION:Results Reserved Words

The following are definitions of the reserved words generated by VISION:Results for this random selection application. The xx in the reserved word name is the application number.

- **DYLSPxRNO**—After the procedure logic statement, IF SAMPLING, application 01 is executed, the field DYLSPO1RNO contains the random number used to determine whether the record is to be selected for inclusion in the sample.
- **DYLSPxCNT**—Starting with zero, each time the procedure logic statement, IF SAMPLING, is executed for application 01, the field DYLSPO1CNT is incremented by one. Therefore, at the end of the run, the field DYLSPO1CNT contains the population size for application 01.
- **DYLSPxPOP**—This is a 5-byte packed field that is automatically generated whenever a random sampling application is defined; xx is the application number. This field can be set by you to the population size before the first execution of an IF SAMPLING xx command; thereafter, the contents of DYLSPxPOP are ignored. DYLSPxPOP must be initialized to a non-zero population size if the population size in the SAMPLE statement is zero.
- **DYLSPxSSZ**—This is a 5-byte packed field that is automatically generated whenever a random sampling application is defined; xx is the application number. This field can be set by you to the sample size before the first execution of an IF SAMPLING xx command; thereafter, the contents of DYLSPxSSZ are ignored. DYLSPxSSZ must be initialized to a non-zero sample size if the sample size in the SAMPLE statement is zero.

Example 2 Randomly Select Records from an Unknown Population

Harvest The Unknown Corporation (HTUC) has an accounts receivable file that contains an unknown number of records, estimated to be from 50,000 to 100,000 in number. You want to randomly select approximately 1% of the population from this file and generate a control listing report of all records selected, as well as write these randomly selected records to an output tape.

First, determine the values to specify in your selection statement. You have decided to use application number 01.

```
SAMPLE 01  RANDOM 1%
```

The keyword **RANDOM** specifies random selection, followed by 1%, the portion of the population you want to select. Below are all the statements required to perform the selection, write selected records to tape, and generate a report of the selected records. Only nine statements are required.

```
1  FILE ACCTREC FB 352 5280
2  BALANCE 5 170 PD 2 E
3  ACCOUNT 7 4 (ACCOUNT NUMBER)
4  NAME 25 85 (ACCOUNT NAME)

5  FILE OACCTREC FB 352 5280 OUTPUT FROM ACCTREC

6  SAMPLE 01 RANDOM 1%

7  ABC: IF SAMPLING 01 NEXT ELSE REJECT ENDIF

8  LIST ACCOUNT NAME BALANCE
9  T1 'SAMPLING LISTING OF INFINITE POPULATION-HTUC'
    WITH 2 AFTER
```

Figure 672 [Example 2](#) Randomly Select Records from an Unknown Population

Statements 1 and 5 define the input and output files, respectively. Statements 2 through 4 define the fields within the accounts receivable file that are to appear on your report (control listing). Statement 7 contains the **IF SAMPLING** command. The application number 01 directly ties this **IF SAMPLING** command to your selection statement, Statement 6. Both the **SAMPLE** definition statement and **IF SAMPLING** command must be present in your program for the selection to be performed.

In the procedure logic, the random number generator is invoked and determines whether the record is to be selected as part of the sample. If the record is to be included, the **TRUE (NEXT)** branch is taken; the record is printed as well as written to the output tape. If the record is not selected, the **FALSE (REJECT)** branch is taken and the record is ignored (neither printed nor written to the output tape).

Statements 8 and 9 define the format of your report listing.

The following shows the Non-Stratified Sampling Summary Report. In addition to the information coded on the simple random statement, this report shows the values for the actual population size, the actual sample size, and the original seed number.

```
1/15/99  VISION:RESULTS NON-STRATIFIED SAMPLING SUMMARY REPORT    PAGE 1

APPLICATION 1 - SIMPLE RANDOM SAMPLING
ESTIMATED POPULATION SIZE                      55,855
SAMPLE PERCENTAGE DESIRED                      1.00000
ACTUAL SAMPLE SIZE                             556
ORIGINAL SEED NUMBER                           79,477
```

Figure 673 [Example 2](#) Output—Randomly Select Records from an Unknown Population

The following is a portion of the records selected in the sample. Each record is printed in the report format, as requested.

SAMPLE LISTING OF INFINITE POPULATION-HTUC

ACCOUNT NUMBER	ACCOUNT NAME	BALANCE
6208657	CHO PYUNG, SUH	32.00
6080669	WILLIAMS, JAMES	24.24
6042325	CHAMBERLINE, FRANCES	34.24
7100035	BROWN, RICHARD	50.62
8033692	MARLETTE, YVETTE	14.95
6114113	NERY, GENEROSO	45.24
6215785	RODEN, HAROLD	170.91
7039085	BLACK, LENORE	50.40
2013568	CULLENDER, EVERRETT J	3,886.33
6058523	CALLEROS, MARCELINO	5.05
6060749	TAYLOR, FLORENCE D	20.00
6046819	YEAGER, CLIFFORD M	15.04
6099467	GOULDING, JOHN	3.00
6113583	BROWN, STUART A	26.20
6090753	BEAVER, OPAL L	15.00
6063977	WALTON, MARVIN L	3.00
6005489	HALL, JOHN H	60.00
7077289	HICKMAN, VERNON	36.95
8033684	WHITEHURST, CHARLES	25.00
6112498	BARRON, BONNIE	36.96

Figure 674 [Example 2](#) Record Sample—Randomly Select Records from an Unknown Population

Chapter 27: Table and Array Handling

You can define up to 255 tables and arrays using a syntax similar to a file definition. The space required for your tables and arrays is dynamically acquired from available memory. You can design a table to hold a number of single entries (one-dimensional), or an array having two to four dimensions.

After the table or array is defined, a few simple commands allow you to store data in it, retrieve the data sequentially or randomly, update an entry, and much more. Table processing is described below. For more information about array processing, see [Syntax for Defining an Array on page 721](#).

Allocating Memory

If you define one or more tables or arrays, VISION:Results automatically acquires additional memory in which to store the tables. The TSIZE parameter in the OPTION statement allows you to control the amount of storage reserved.

For example, if you need 2K for the entries in one table and 4K for another, code the following to allocate enough space for both tables:

```
OPTION TSIZE 6K
```

Figure 675 2K is Needed for One Table and 4K for Another

You can use the TABLE/ARRAY processor of VISION:Results to create and access tables and search large amounts of data.

Effective with r6, table storage is always allocated above the 16M line. Parameter setting for DYLINSTL TABLHI and OPTION statement TABAREA ABOVE or BELOW are ignored.

The guidelines for deciding how much memory to allow are:

- Round the number of bytes of data you expect to store in a given table up to the next multiple of 1024 (1K).
- Be generous in your estimate unless you know exactly how much data must be tabled. Remember that many kinds of tables tend to grow.

- Variable-length tables require an additional 2 bytes per entry to hold the entry length.
- Arrays always require the maximum space (the product of the element length and all the dimensions).

If sufficient storage is not available to satisfy the memory requested, an 80A system abend or a 602 completion code (VSE) results. If this happens, increase your partition size or decrease your TSIZE.

The minimum TSIZE allowed is 2K; the maximum is 63M or 63000K. The maximum size of any 1 table is 32M. If TSIZE is not specified, the default of 20K is used.

Working with Tables

This section describes how to perform various tasks with tables.

Syntax for Defining a Table

```
TABLE tablename STATUS dataname
  {F entrylen | V maxlen LENGTH dataname}
  {ENTRY | OFFSET} dataname
  [KEYLOC {dataname | n} KEYLEN {dataname | n}]
  [FROM filename]
  [{RETAIN | NORETAIN}]
Table entry field definition(s)
```

Figure 676 Syntax for Defining a Table

tablename	Required. 1- to 8-character alphanumeric table name used in all accesses to this table.
STATUS	Required. Must be followed by a self-defining data name of your choice. This data name is 1 byte in length and contains one of the following values after any table handling command: Y The requested operation completed successfully. N The requested operation is unsuccessful. L The requested operation is unsuccessful because the contents of the LENGTH dataname were invalid. See the sections on handling variable-length tables for more information.
F entrylen	Required for fixed-length tables. Entrylen is the numeric length of each entry and must be between 1 and 32767, inclusive.
V maxlen	Required for variable-length tables. Maxlen is the numeric maximum entry length and must be between 1 and 32765, inclusive.

LENGTH	Required for variable-length tables. Must be followed by a self-defining data name of your choice. This data name passes the length of a given entry to or from VISION:Results.
ENTRY	Required for fixed-length tables. Must be followed by a self-defining data name of your choice. This data name passes an entry number to or from VISION:Results.
OFFSET	Required for variable-length tables. Must be followed by a self-defining data name of your choice. This data name passes the offset or relative byte address (relative to 0) of an entry to or from VISION:Results.
KEYLOC	Key location. Required if SEARCH, BINSEARCH, or TSORT is used in procedure logic for this table. It must be followed by a self-defining data name or a numeric literal. Allowable values are 1 to 32765, inclusive.
KEYLEN	Key length. Required if SEARCH, BINSEARCH, or TSORT is used in procedure logic for this table. It must be followed by a self-defining data name or a numeric literal.
	Allowable values are 1 to 256, inclusive. The key specified by KEYLOC and KEYLEN must be fully contained within the shortest table entry, or the results of your SEARCH, BINSEARCH, or TSORT are unpredictable.
FROM filename	This option causes the named file to be read in its entirety and loaded into the table before control is passed to your procedure logic. Be sure there is at least one other input file defined, or your program could end as soon as the table is loaded.
RETAIN NORETAIN	This option specifies that this table and its contents are either passed to a subsequent multiple-report request or deleted at the end of the current request. A retained table requires neither JCL nor the use of an external medium.

The following rules govern automatic table loading:

- The source file must have the same record format (F or V) as the table, and the FILE statement must include the record length.
- The file must be defined as sequential input and must not use the SKIP, IO, EXIT, or MODIFY keywords.
- For fixed-length entry tables, if the file record length differs from the table entry length, the record is truncated or padded to the right with blanks, as necessary, as each table entry is built.
- For variable-length tables, the maximum file record length must not be greater than the maximum table entry length.

Table Entry Field Definitions

Code one or more field definitions following the TABLE statement to assign a name to the table entry and to any fields within it. The syntax of a table entry field definition is the same as that of any other data name.

A data name having the same name as the table is automatically generated by VISION:Results. It is a 1-byte character field whose location is the first byte of the table entry definition.

Loading a Table

Fixed-Length Entries

Define your table as follows:

```
TABLE DEPTTB F 33 ENTRY TBLENT STATUS TBSTAT  
  DEPTNUM 3 PD  
  DEPTNAME 30
```

Figure 677 Define Table for Fixed Length

Code procedure logic to read records from a file, build a table entry, and store it:

```
TBLOAD:  
  READ DEPTMSTR  
  IF EOFFLAG EQ 'E' GOTO ENDLOAD ENDIF  
  MOVE MSTRNUM TO DEPTNUM  
  MOVE MSTRNAME TO DEPTNAME  
  STORE DEPTTB  
  IF TBSTAT NE 'Y' GOTO TBLFULL ENDIF  
  GOTO TBLOAD  
ENDLOAD:
```

Figure 678 Code Procedure Logic for Fixed Length

Before the STORE, the ENTRY dataname contains the ordinal number of the last entry stored. The STORE command causes the contents of the table entry area to be added to the table as the next entry. The STATUS area data name (TBSTAT) is checked after every STORE to ensure that there is enough room in the tabling area to hold the new entry.

After the STORE, the ENTRY dataname contains the ordinal number of the entry just stored.

If the first 33 bytes of each record on the DEPTMSTR file match the format of the table entry and no record selection is required, you can request an automatic table load instead. For more information, see [FROM filename on page 707](#).

This form of STORE results in the end-of-table marker being reset.

Variable-Length Entries

Define your table as follows:

```
TABLE VARTBL V 209 LENGTH ENTLEN OFFSET RBA STATUS TBSTAT  
  TACCTNO 9  
  TBACKITEMS 200 ; UP TO 10 ITEMS @ 20 BYTES EACH
```

Figure 679 Define Table for Variable Length

You can then load the table as follows:

```
TBLOAD:
  READ ACCTFILE
  IF EOFFLAG EQ 'E' GOTO ENDLOAD ENDIF
  MOVE ACCTNO TO TACCTNO
  LGTH = ITEMS * 20          ; CALC LENGTH OF BACKITEMS
  IF LGTH GT 200 GOTO TOOMANY ENDIF
  MOVE BACKITEMS TO TBACKITEMS LENGTH LGTH
  ENTLEN = LGTH + 9          ; SET LENGTH TO STORE
  STORE VARTBL
  IF TBSTAT NE 'Y' GOTO TBLFULL ENDIF
  GOTO TBLOAD
ENDLOAD:
```

Figure 680 Code Procedure Logic for Variable Length

Before the STORE, the OFFSET dataname contains the offset into the table at which to store the data.

The STORE command causes ENTLEN bytes of data (in the example) to be moved from the table entry area into the table at the supplied offset. The STATUS dataname (TBSTAT) is checked after every STORE to ensure that there was enough room in the tabling area to hold the new data.

After the STORE, the OFFSET dataname contains the next available offset.

If each table entry is to hold one complete input record and no record selection is required, you can request an automatic table load instead. For more information, see [FROM filename on page 707](#).

This form of STORE results in the end-of-table marker being reset.

Sequential Retrieval

Fixed-Length Entries

Once your table is loaded, reset the ENTRY dataname to 1 or any other starting entry number. For example:

```
MOVE 1 TO ENTNUM
```

Figure 681 ENTRY dataname Reset

Then, issue a RETRIEVE and check that the entry was found:

```
RETRIEVE tablename
IF TBSTAT EQ 'Y' GOTO ENTRYFOUND ENDIF
```

Figure 682 RETRIEVE Fixed Length

The RETRIEVE command returns the requested entry to the area following your TABLE statement and increments the ENTRY dataname by 1. The next RETRIEVE obtains the next entry and so on, until the entries are exhausted.

When the ENTRY dataname contains a value higher than the number of entries in the table, the table entry area remains unchanged after a RETRIEVE and the STATUS dataname contains an N.

Variable-Length Entries

Once your table has been loaded, reset the OFFSET dataname to 0. For example:

```
MOVE 0 TO RBA
```

Figure 683 Reset OFFSET dataname

Then, issue a RETRIEVE and check that the entry was obtained:

```
RETRIEVE tablename  
IF TBSTAT EQ 'Y' GOTO ENTRYFOUND ENDIF
```

Figure 684 RETRIEVE Variable Length

The RETRIEVE command returns the (first) entry to the area following your TABLE statement, returns the length of the entry in the LENGTH dataname, and increments the OFFSET dataname to point to the next table entry. Therefore, the next RETRIEVE obtains the next entry and its length, and so on, until the entries are exhausted.

When the OFFSET dataname contains a value higher than the end of the table, the table entry area and the LENGTH dataname remain unchanged after a RETRIEVE and the STATUS dataname contains an N.

To do a sequential retrieval of variable-length entries from some starting point other than the beginning, you need to supply the OFFSET at which to start. This offset must be saved by you from some prior table processing because it is not generally possible to calculate it.

Sequential Search

The SEARCH operation scans the entries in a table, serially from a given starting point, looking for an entry whose key matches the search argument. The TABLE statement defining a table to be searched must include the KEYLOC and KEYLEN parameters. For more information, see [Syntax for Defining a Table on page 706](#).

Fixed-Length Entries

Before the table can be searched, you must move the entry number of the first entry to be included in the search into the ENTRY dataname. In addition, if either the KEYLOC or KEYLEN parameters in the TABLE statement are not coded using an absolute value, the actual KEYLOC and/or KEYLEN values must be set in the related data names at some point prior to the SEARCH.

To search the table, code:

```
SEARCH tablename argument-name
```

Figure 685 Search the Fixed-Length Table

Argument-name can be the data name of any field in your program that contains the value for which you are searching. VISION:Results does a character comparison of the argument with each entry, using the number of bytes defined by the KEYLEN parameter, until a match is found or the end of the table is reached.

If a matching entry is found, the entry is moved to the area following your TABLE statement, its entry number is returned in the ENTRY dataname, and a Y is returned in the STATUS dataname.

If a matching entry is not found, the ENTRY dataname is unchanged and the STATUS dataname contains an N. The contents of the table entry area are unpredictable when no match is found.

To resume the search after a matching entry has been found, be sure to add 1 to the ENTRY dataname before executing the SEARCH verb again. Failing to do this results in a never-ending loop because the same entry is returned every time.

Variable-Length Entries

Before the table can be searched, you must move the offset (usually 0) of the first entry to be included in the search into the OFFSET dataname. In addition, if either the KEYLOC or KEYLEN parameters in the TABLE statement are not coded using an absolute value, the actual KEYLOC and/or KEYLEN values must be set in the related data names at some point prior to the SEARCH.

To search the table, code:

```
SEARCH tablename argument-name
```

Figure 686 Search the Variable-Length Table

The argument-name can be the data name of any field in your program that contains the value for which you are searching. VISION:Results does a character comparison of the argument with each entry, using the number of bytes defined by the KEYLEN parameter, until a match is found or the end of the table is reached.

If a matching entry is found, the entry is moved to the area following your TABLE statement, its length is returned in the LENGTH dataname, its offset is returned in the OFFSET dataname, and a Y is returned in the STATUS dataname.

If a matching entry is not found, the LENGTH and OFFSET data names are unchanged and the STATUS dataname contains an N. The contents of the table entry area are unpredictable when no match is found.

To resume the search after a matching entry has been found, calculate the OFFSET dataname as: the current OFFSET plus the length of the entry just found plus 2 for the length attribute field.

`RBA = RBA + ENTLEN + 2`

Figure 687 OFFSET dataname

Failing to do this results in a never-ending loop because the same entry is returned every time.

Binary Search

For tables containing at least 20 entries, a binary search is substantially faster than a sequential search at finding one unique entry. For very large tables, the speed is especially evident. For example, a table of 1,000 entries requires a maximum of ten comparisons to find a given entry, a table of 2,000 entries requires up to 11 compares, a table of 4,000 entries requires up to 12 compares, and so on.

You can use a binary search if your table meets the following conditions:

- The entries are fixed length.
- The entries are stored in ascending order by key.

Both the KEYLOC and the KEYLEN parameters must be coded in your TABLE statement. For more information, see [Syntax for Defining a Table on page 706](#). If either the KEYLOC or KEYLEN parameters are not coded using an absolute value, the actual KEYLOC or KEYLEN values must be set in the related data names at some point prior to the SEARCH.

To request a binary search, code:

`BINSEARCH tablename argument-name`

Figure 688 Request a Binary Search

The argument-name is the data name of any field in your program that contains the value for which you are searching. VISION:Results does a character comparison of the argument with each entry, using the number of bytes defined by the KEYLEN parameter, until a match is found or the end of the table is reached.

If a matching entry is found, the entry is moved to the area following your TABLE statement, its entry number is returned in the ENTRY dataname, and a Y is returned in the STATUS dataname.

If a matching entry is not found, the ENTRY dataname is unchanged and the STATUS dataname contains an N. The contents of the table entry area are unpredictable when no match is found.

Sorting

VISION:Results provides a sort facility that can arrange table entries into ascending or descending order by key. The following conditions must be met:

- The entries are fixed length.
- The entry length does not exceed 256 bytes.

This sort is recommended for relatively small tables. A noticeable increase in CPU time usage occurs if a table containing more than about 700 entries is sorted. The overall performance of the sort also depends on entry length and key length. As a rule, the shorter the lengths, the faster the sort runs.

Both the KEYLOC and the KEYLEN parameters must be coded in your TABLE statement. See the TABLE statement syntax for more information. If either the KEYLOC or KEYLEN parameters are not coded using an absolute value, the actual KEYLOC/KEYLEN values must be set in the related data names at some point prior to the sort.

To request the sorting of a TABLE, code:

```
TSORT tablename [{ASCENDING | A | DESCENDING | D}]
```

Figure 689 Request the Sorting of a TABLE

If no sequence option follows the table name, ASCENDING is assumed. There is no limit to the number of TSORTs you can perform in a given program.

Random Retrieval

Fixed-Length Entries

You can randomly retrieve any entry in a table if you know its entry number. The ENTRY dataname contains the number of the entry just stored or retrieved after a successful STORE, SEARCH, or BINSEARCH operation. It is also possible that you know the entry number because it bears a direct relationship to some other variable (such as a table containing one entry for each month in the year). Move the number of the entry you want to the ENTRY dataname; then code:

```
RETRIEVE tablename RANDOM
```

Figure 690 Randomly Retrieve Entry in a Table

VISION:Results attempts to acquire the entry whose number is in the ENTRY dataname. If successful, the entry is returned to the table entry area, following your TABLE statement, and a Y is placed in the STATUS dataname. If unsuccessful, an N is moved to the STATUS dataname and the table entry area remains unchanged. The only reason for not finding an entry is that the table contains fewer entries than the ordinal number requested.

Variable-Length Entries

You can randomly retrieve any entry in a table if you know its offset. The offset of the first entry in the table is 0. After a successful SEARCH operation, the OFFSET dataname contains the offset of the entry just retrieved.

Just before a STORE operation, the OFFSET dataname contains the offset of the entry about to be stored. Move the offset of the entry you want to the OFFSET dataname, then code:

```
RETRIEVE tablename RANDOM
```

Note: If the offset is within the table but does not point to an entry correctly, the result of the RETRIEVE is unpredictable.

VISION:Results attempts to acquire the entry whose offset is in the OFFSET dataname. If the requested offset is within the table, the entry is returned to the table entry area (following your TABLE statement), its length is placed in the LENGTH dataname, and a Y is returned in the STATUS dataname. If not, an N is moved to the STATUS dataname, and the table entry area and the LENGTH dataname remain unchanged.

Random Replacement

Fixed-Length Entries

Any entry in a table can be replaced (updated) randomly if you know its entry number. The ENTRY dataname contains the following:

- The number of the entry about to be obtained before a RETRIEVE operation.
- The entry number found after a successful SEARCH or BINSEARCH.

It is also possible that you know the entry number because it bears a direct relationship to some other variable, such as a table containing one entry for each month in the year. Move the number of the entry to be updated to the ENTRY dataname, then code:

```
STORE tablename RANDOM
```

VISION:Results replaces the entry whose number is in the ENTRY dataname with the contents of the table entry area, following your TABLE statement, unless the requested entry is outside of the table. If successful, a Y is returned in the STATUS dataname; if not, an N is returned.

The end-of-table marker is not changed when the keyword RANDOM is specified. Specify RANDOM; otherwise, the end-of-table marker is reset. For more information, see [Loading a Table on page 708](#).

Variable-Length Entries

Any entry in a table can be replaced (updated) randomly if the following conditions are true:

- The last tabling function against this table is a RETRIEVE or SEARCH and successfully obtained this entry.
- You have not modified the contents of the LENGTH dataname as returned by the last RETRIEVE/SEARCH operation for this table.

```
STORE tablename RANDOM
```

Figure 691 Random Replacement in a Table with Variable Length

If the random store is, in fact, at the same offset and for the same length as the last entry obtained from this table, that entry is replaced with the contents of the table entry area, following your TABLE statement, and a Y is returned in the STATUS dataname.

If you attempt to replace an entry that is not the last one retrieved or you have modified the contents of either the OFFSET or LENGTH data names replacement does not occur, and an N is placed in the STATUS dataname.

The end-of-table marker is not changed when the keyword RANDOM is specified. Specify RANDOM; otherwise, the end-of-table marker is reset. For more information, see [Loading a Table on page 708](#).

Deleting a Table

If you want to delete a table during processing, you can do it in either of the following ways:

- [Physical DELETE](#)
- [Logical Deletion](#)

Each method has certain advantages and disadvantages. Select the one that best suits a given situation.

Physical DELETE

All storage being held by the table is returned to free space, for use by other tables, except for 32 bytes per 4K of table that is not recovered. The first subsequent STORE to the same table name causes a new table, containing the one entry, to be created.

You can store additional entries in the table, issue other tabling commands against it, then delete it again if you want, and so on. However, every DELETE of an existing table results in the loss of at least 32 bytes. For example, if the table is being used for the temporary storage of one batch of records and there are 1,000 batches, the repeated deletion of a table would eventually use up the tabling area.

DELETE is best used when you are finished with a given table and want to reuse the space for a different table. The syntax of the DELETE command is:

```
DELETE tablename
```

Figure 692 Delete Command—Tables

Logical Deletion

In this method, you reset the end-of-table marker by deliberately storing an entry on top of the existing first entry without using the RANDOM keyword. If you want, additional entries can be stored in the normal way. The previous contents of the table are ignored and the space currently allocated to the table is reused.

You can even expand the table beyond its original size; however, you cannot make it smaller unless you issue a DELETE. The space can be reused over and over again without impacting the total available tabling area.

For fixed-length entry tables, reset the ENTRY dataname to 0, move the wanted values to the table entry area, and issue a STORE:

```
MOVE 0 TO ENTNUM
      .
      .MOVE values
      .
STORE tablename
```

Figure 693 Logical Deletion of Fixed-Length Entry Tables

This technique stores the contents of the table entry area as entry #1 in an existing table.

For variable-length entry tables, reset the OFFSET dataname to 0, move the desired values to the table entry area, and issue a STORE:

```
MOVE 0 TO RBAVAL
      .
      .MOVE values
      .
STORE tablename
```

Figure 694 Logical Deletion of Variable-Length Entry Tables

This technique stores the contents of the table entry area, using the value in the LENGTH dataname for the length, as the first entry in an existing table.

USE Command

This command is coded in a secondary request to make the contents of a previously defined and retained table available in this request. The syntax of the USE command is:

```
USE tablename [NORETAIN]
```

Figure 695 USE Command—Tables

The contents of the TABLE statement referred to, as well as all table field definitions subordinate to it, become a part of this secondary request. New table field definitions can be defined after the USE command, but they are not retained for any subsequent requests. The contents of the table itself are the same as the last time this table was referenced.

If NORETAIN is coded, the table is deleted at the end of this request; otherwise, it is kept. When all RETAINED tables and arrays have been used with NORETAIN, the tabling area is freed to the operating system. Subsequent use of a table causes a new tabling area to be obtained.

Error Return Codes

Your VISION:Results program abends with a user return code, as follows, if the table handler uses up all the available storage before completing certain automatic functions:

User 600	Insufficient storage for the requested automatic table load function (using the FROM keyword).

The allocated table area does not hold the entire input file. In this case, the probable solution is to increase the value of the TSIZE parameter on your OPTION statement. See [Allocating Memory on page 705](#). Another solution is to decrease the size of the entries being stored.

When the FROM keyword is not used, your VISION:Results program provides a return code in the first two bytes of DYLCOMRG. You must also specify the RETCODE parameter in your DYLINSTL installation macro. (For more information about DYLINSTL, see the *Advantage VISION:Results for z/OS Installation Guide*.)

To examine the return code, you code the following program logic:

```
WORKAREA
  XX 2
  REDEFINE XX
  YY 2 BI
  command tablename
  MOVE DYLCOMRG TO XX
  IF YY NE 0
    handle non-zero return-code
  ENDIF
```

The return codes are:

00	Successful operation
01	End of table
02	Record not found
03	Premature end of table
04	New table already in use
05	Cannot search free-form table (or VSE unsuccessful GETVIS)
06	Invalid entry length The defined entry LENGTH field for the table contains a value less than 1 or greater than 32,767.
07	Requested table not found
08	Table area is full The TSIZE specified must be increased, or the maximum size of 32M has been reached.
09	Invalid or missing parameter
0A	Not fixed length records
0B	LRECL too large to sort

User VSE only. Insufficient memory to satisfy the TSIZE requirement.
602

Table Examples

Example 1 Binary Search

A table is loaded automatically with the first 31 bytes of each record on a customer master file. A binary search of the table is performed to obtain the customer's name for a sales report line.

Note: **Bold** areas are required for VSE systems.

```
OPTION TSIZE 150K XREFA

FILE SALES TAPE SYS035 FB 128 1280
  SALESMAN 3 3
  CUSTNUM 6 26
  PRODUCT 7 32
  QUANTITY 3 54 PD Z
  UNITPRICE 4 81 PD 2 E

FILE CUSMSTR DISK 3350 FB 140 2800
  MCUSTNO 6 1
  MCUSTNAM 25 7
  FILLER 109

TABLE CUSTBL F 31 FROM CUSMSTR STATUS FOUND
```

Figure 696 Example 1 Binary Search (Page 1 of 2)

```

        ENTRY TBLENT KEYLOC 1 KEYLEN 6
        CNUMBER 6
        CNAME 25 (CUSTOMER NAME)

WORKAREA
    TOT 7 PD 2 E
    TALLY E 3

    IF CUSTNUM EQ CNUMBER GOTO CONTINUE ENDIF
    BINSEARCH CUSTBL CUSTNUM
    IF FOUND NE 'Y'
        MOVE '*** NOT ON FILE ***' TO CNAME
    ENDIF
    MOVE CUSTNUM TO CNUMBER

CONTINUE:
    TOT = QUANTITY * UNITPRICE
    LIST SUPPRESS (SALESMAN CNAME) PRODUCT
    QUANTITY TOT

CONTROL SALESMAN
ON CHANGE IN SALESMAN
    LIST 'TOTAL SALESMAN' AT CNAME TALLY SUM TOT
    WITH 2 BEFORE AND EJECT AFTER
ON FINAL
    LIST '** FINAL TOTAL **' AT PRODUCT SUM TOT

T1 'S A L E S   B Y   S A L E S M A N' WITH 2 AFTER
T1+100 DYLDATEPAG

```

Figure 696 Example 1 Binary Search (Page 2 of 2)

Example 2 Sort a Small Variable-Length Table Using TSORT

Variable-length table entries need to be accessed in a new sequence. As each table entry is stored, the key value and offset of the entry are stored in a fixed-length entry table.

This second table is sorted using TSORT. The sorted entries are then used to randomly retrieve the variable-length entries in the required sequence.

```

OPTION TSIZE 50K XREFA

FILE VARFILE ESDS V 50 LENGTH INLEN
    RKEY 6 18
    RCODE 1 41

TABLE VTABLE V 50 LENGTH VTBLN STATUS VTBSTAT OFFSET VTBADDR
    VTREC 50

TABLE TAGTABLE F 10 STATUS TTBSTAT ENTRY TTBENT
    KEYLOC 5 KEYLEN 6
    TTADDR 4 BI
    TTKEY 6

    IF RCODE EQ '1' '6' '7' 'X' 'P'
        REJECT ENDIF
    MOVE VARFILE TO VTREC LENGTH INLEN
    MOVE INLEN TO VTBLN
    MOVE RKEY TO TTKEY
    MOVE VTBADDR TO TTADDR
    STORE VTABLE
    IF VTBSTAT NE 'Y'
        PRINT 'OUT OF SPACE (T1)'
        MOVE 101 TO DYLDRETURN STOP ENDIF
    STORE TAGTABLE
    IF TTBSTAT NE 'Y'
        PRINT 'OUT OF SPACE (T2)'

```

Figure 697 Example 2 Sort a Small Variable-Length Table Using TSORT (Page 1 of 2)

```
MOVE 101 TO DYLRRETURN STOP ENDIF

ON END OF INPUT
TSORT TAGTABLE
MOVE 1 TO TTBENT

NXT VREC:
RETRIEVE TAGTABLE
IF TTBSTAT NE 'Y'
PRINT 'DONE' STOP ENDIF
MOVE TTADDR TO VTBADDR
RETRIEVE VTABLE RANDOM
PRINT VTREC LENGTH VTBLN
GOTO NXT_VREC
```

Figure 697 Example 2 Sort a Small Variable-Length Table Using TSORT
(Page 2 of 2)

Example 3 Serial Search, Add New Entry If Not Found

For each input record, if the charge code is found in the table, a count field is increased by 1 and the updated entry is replaced in the table.

If the charge code is not found, a new 'last' entry is added to the table, and its entry number is saved. When all input charge codes have been accounted for, the table entries are sorted and a report is printed.

The DUMMY file keeps the job from ending automatically when the only (other) input file reaches end of file.

Bold text is required for VSE systems.

```
OPTION XREF
FILE CHARGES FB 97 9700 TAPE SYS010 STATUS EOFCHG
CHGCODE 4 14
FILE XYZ DUMMY

TABLE CHGTABLE F 8 STATUS TBSTAT ENTRY ENTNUM
KEYLOC 1 KEYLEN 4
TBREC 8
TBCHGCD 4 1 (CHARGE CODE)
TBCHGCNT 4 PD (COUNT) RIGHT

WORKAREA
LASTENT 4 BI VALUE 0
EOT 1
REPORT 30 WIDE
ENTNUM=1
IF EOFCHG EQ 'E' GOTO SORT_PRT ENDIF
SEARCH CHGTABLE CHGCODE
IF TBSTAT EQ 'Y'
TBCHGCNT = TBCHGCNT + 1
STORE CHGTABLE RANDOM
REJECT ENDIF
MOVE CHGCODE TO TBCHGCD
MOVE DYLONG TO TBCHGCNT
MOVE LASTENT TO ENTNUM
STORE CHGTABLE
IF TBSTAT NE 'Y'
PRINT '**ERROR- TABLE FULL'
MOVE 301 TO DYLRRETURN STOP ENDIF
MOVE ENTNUM TO LASTENT
REJECT
```

Figure 698 Example 3 Serial Search, Add New Entry If Not Found (Page 1 of 2)


```
SORT_PRT:
  RETRIEVE CHGTABLE
  IF TBSTAT NE 'Y'
    MOVE 'E' TO EOT GOTO SORTRN ENDIF
  RELSORT
  GOTO SORT_PRT
SORTRN:
  SORT TBREC USING TBCHGCD UNTIL EOT
  LIST TBCHGCD TBCHGCNT

ON FINAL
  LIST 'TOTAL CHARGES' AT TBCHGCD SUM TBCHGCNT
T1 ' '
T1+1 DYLDATE
T1+18 DYLPAGE
```

Figure 698 Example 3 Serial Search, Add New Entry If Not Found (Page 2 of 2)

Working with Arrays

This section describes how to perform various tasks with arrays.

Syntax for Defining an Array

```
ARRAY arrayname STATUS dataname
  F element-length
  {DIMENSION | DIM} (n1 n2 [n3] [n4])
  [FILL fillchar]
  [{RETAIN | NORETAIN}]
Array element field definition(s)
```

Figure 699 Syntax for Defining an Array

arrayname	Required. The array name is a 1- to 8-character alphanumeric name used in all accesses to this array.
STATUS	Required. Must be followed by a self-defining data name of your choice. This data name is 1 byte in length and contains one of the following values after any array handling command: Y The requested operation completed successfully. N The requested operation was unsuccessful.
F element-length	Required. F because each element in an array must be the same length. The element-length is the numeric length of each element and must be between 1 and 32767, inclusive.

DIMENSION DIM	Required. Defines the size of each dimension in the array. Must be followed by two to four numeric literals enclosed in parentheses. The number of literals defines the number of dimensions. Each dimension can be in the range of 1 to 999999, inclusive. To use commas to separate the literals, each comma must be followed by at least one space (for example, 5, 12, 6).
FILL fillchar	Optional. The fillchar is a 1-character alphanumeric or hex literal that specifies the character to be replicated throughout every element in the array during initialization. If FILL x is not coded, the default fill character is a blank.
RETAIN NORETAIN	Optional. Specifies that this array and its contents are to be passed to a subsequent multiple report request, or deleted at the end of the current request. A retained array requires neither JCL nor the use of an external medium.
Array Element Field Definitions	One or more VISION:Results field definitions should be coded, following the ARRAY statement, to assign a name to the array element and to any fields within it. The syntax of an array element field definition is the same as that of any other data name. A data name having the same name as the array is automatically generated by VISION:Results. It is a 1-byte character field whose location is the first byte of the array element definition.

Loading an Array

For example, an array can be defined as follows:

```
ARRAY HISTORY F 30 STATUS HISTAT DIM(10, 12)
  HISTREC 30
```

Figure 700 Loading an Array

The entire 120 elements are formatted prior to executing your first procedural statement. Any element in the array can be replaced or updated, as follows:

```
MOVE INREC TO HISTREC
STORE HISTORY (INYEAR INMONTH)
```

INYEAR is any numeric, packed, or binary field whose value is between 1 and 10, and INMONTH is a similar field with a value between 1 and 12.

The number of coordinates in parentheses must be the same as the number of dimensions defined in the ARRAY statement. The actual value of a given coordinate, when the STORE is executed, must not be less than 1 or greater than its related dimension size. However, no checking is performed by VISION:Results to ensure that the value of a given coordinate is within its limits.

If the requested element is outside of the array, an N is returned in the STATUS dataname.

Retrieving from an Array

All entries in an array are retrieved in the same way:

```
RETRIEVE arrayname (coordinates)
```

Figure 701 Retrieving from an Array

The coordinates stand for two to four data names, each of which has a data type of NU, PD, or BI. You must supply one data name for each dimension defined in the DIM parameter on the ARRAY statement. For example:

```
RETRIEVE HISTORY (YEAR MONTH)
```

Figure 702 Array Data Name

Each coordinate must contain a value between 1 and the related dimension size. However, no checking is performed to ensure that the value of a given coordinate is within its limits.

The RETRIEVE command returns the requested element into the area following your ARRAY statement. If the requested element is outside of the array, an N is returned in the STATUS dataname.

Deleting an Array

All of the storage allocated to an array can be returned to free space for use by other tables by coding:

```
DELETE arrayname
```

Figure 703 Deleting an Array

Once an array is deleted, it cannot be reinstated or reused.

USE Command

This command is coded in a secondary request to make the contents of a previously defined and retained array available in this request. Its syntax is as follows:

```
USE arrayname [NORETAIN]
```

Figure 704 USE Command—Array

The contents of the ARRAY statement referred to, as well as all array field definitions subordinate to it, become a part of this secondary request. New array field definitions can be defined after the USE command, but they are *not* retained for any subsequent requests. The contents of the array itself are the same as they were the last time this array was referenced.

If NORETAIN is coded, the array is deleted at the end of this request; otherwise, it is kept. When all retained tables and arrays have been used with NORETAIN, the tabling area is freed to the operating system. Subsequent use of an array causes a new tabling area to be obtained.

Error Return Codes

Your VISION:Results program abends with a user return code, as follows, if the table handler uses up all the available storage before completing certain automatic functions:

User 601 Insufficient storage to hold your array.

The number of entries in an array is the product of all dimensions. There is not enough room to hold that many entries.

In this case, increase the value of the TSIZE parameter on your OPTION statement. See [Allocating Memory on page 705](#). Another solution is to decrease the size of the entries being stored, or to decrease the size of the dimensions of your array.

When the FROM keyword is not used, your VISION:Results program provides a return code in the third and fourth bytes of DYLCOMRG. You must also specify the RETCODE parameter in your DYLINSTL installation macro. (For more information about DYLINSTL, see the *Advantage VISION:Results for z/OS Installation Guide*.)

To examine the return code, you code the following program logic:

```
WORKAREA
  RETCODE      4
  YY           2  3  BI
command arrayname
MOVE DYLCOMRG TO XX
IF YY NE 0
  handle non-zero return-code
ENDIF
```

The return codes are:

00	Successful operation
01	End of array
02	Record not found
03	Premature end of array
04	New array already in use
05	Cannot search free-form array (or VSE unsuccessful GETVIS)
06	Invalid entry length The defined entry LENGTH field for the array contains a value less than 1 or greater than 32,767.
07	Requested array not found

08 Array area is full
 The TSIZE specified must be increased, or the maximum size of
 32M has been reached.

09 Invalid or missing parameter

0A Not fixed length records

0B LRECL too large to sort

User 602 VSE only. Insufficient memory to satisfy the TSIZE requirement.

Array Example

This program example tells VISION:Results to load an array by column and retrieve it by row, as in a telephone directory.

An array is loaded by column, with employee name and extension information. When the array is full, it is printed by row. When the input file reaches the end, the unused portion of the array is cleared and the final page is printed.

```
OPTION TSIZE 5K XREFA

FILE EMPLMST FB 240 7200 DISK 3350
  EMPLNAME  25  15
  BILLED_AMT 4   91

ARRAY EMPLIST F 29 DIM (50 3) STATUS TBLSTAT
  ANAME 25
  AEXT  4

WORKAREA
  ROW    2 PD VALUE 1
  COLUMN 1 PD VALUE 1

WORKAREA
  PRTLINE 101
  NAME1   25  1
  EXT1    4 28

REPORT 102 WIDE

  MOVE EMPLNAME TO ANAME
  MOVE BILLED AMT TO AEXT
  STORE EMPLIST (ROW COLUMN)
  ROW = ROW + 1
  IF ROW GT 50
    ROW = 1
    COLUMN = COLUMN + 1 ENDIF
  IF COLUMN GT 3 GOTO PRT_PAGE ENDIF
  REJECT

PRT_PAGE:
  COLUMN = 1
  INX = 0
```

Figure 705 Array Example (Page 1 of 2)

```
BLD_LINE:
  RETRIEVE EMPLIST (ROW COLUMN)
  MOVE ANAME TO NAME1 (INX)
  MOVE AEXT TO EXT1 (INX)
  COLUMN = COLUMN + 1
  IF COLUMN LE 3
    INX = INX + 35
    GOTO BLD_LINE ENDIF
PRT_LINE:
  LIST PRTLINE
  ROW = ROW + 1
  IF ROW LE 50 GOTO PRT_PAGE ENDIF
  ROW = 1
  COLUMN = 1

PRT_EX:
  LIST ' ' WITH EJECT AFTER
  REJECT

ON END OF INPUT
  IF ROW EQ 1 AND COLUMN EQ 1 STOP ENDIF
  MOVE SPACE TO ANAME
  MOVE SPACE TO AEXT

CLR_REST:
  STORE EMPLIST (ROW COLUMN)
  ROW = ROW + 1
  IF ROW GT 50
    ROW = 1
    COLUMN = COLUMN + 1 ENDIF
  IF COLUMN LE 3 GOTO CLR_REST ENDIF
  PERFORM PRT_PAGE TO PRT_EX

T1 ' ' WITH 2 AFTER
T1+1 'NAME' T1+28 'EXT'
T1+36 'NAME' T1+63 'EXT'
T1+71 'NAME' T1+98 'EXT'
```

Figure 705 Array Example (Page 2 of 2)

Chapter 28: Using Letter Writing

This chapter describes the VISION:Results letter writing capabilities for standard impact and laser printers. Several examples using the letter writing function are presented in actual use.

Using the Letter Writing Function

VISION:Results produces documents that can contain variable information such as the names and addresses of the various recipients. Some of the items you can generate include:

- Form letters
- Confirmation notices
- Inquiries
- Contract and lease agreements
- Control lists
- Summaries
- Memoranda
- Two-up letters
- Address labels

The letter writing function is useful with systems such as accounts payable, accounts receivable, bank accounts, mortgages, securities, and contract administration.

Some of the features include:

- Letters can be generated in two-up format.
- Letters that can be directed to specific letter files.
- Each letter's print line can be routed through an exit subroutine.
- Letters can be up to 260 characters wide.
- ASA carriage control characters can be specified for additional print control.
- The first line of a paragraph can be indented.
- An entire paragraph can be indented.

Implementing Letter Writing

[Figure 706 on page 728](#) shows a sample letter produced using VISION:Results, and [Figure 707 on page 728](#) shows the required program statements.

```
NOVEMBER 30, 1998

ERNESTO TORRES
23444 PARK LANE
LOS ANGELES, CA

DEAR ERNESTO TORRES:

YOUR INTEREST IN VISITING OUR STATE IS AS IMPORTANT TO US AS IT IS TO YOU.

OPPORTUNITIES FOR TRAVEL AND TOURISM TO AND WITHIN AMERICA'S SUNSHINE STATE ARE GREAT B
UT WE WANT THEM TO BE EVEN BETTER AND MORE AVAILABLE TO OUR VISITORS. INFORMATION ABOU
T YOUR TRAVEL HELPS US TO PLAN AN EVEN MORE EXCITING STAY THE NEXT TIME YOU VISIT OUR S
TATE.

VERY TRULY YOURS,

FLORIDA STATE DEPARTMENT OF TOURISM

ENCLOSURES: TRAVEL BROCHURES
```

Figure 706 Sample Letter

```
1  FILE ARFILE  FB
2      LASTNAME 15  11
3      FIRSTNAME 10 26
4      ADD1      25 110
5      ADD2      25 135

6  LTH 10 75 WIDE

7  LTD @(ASIS)NOVEMBER 30, 1998
8  LTD @(SP3) @FIRSTNAME @LASTNAME
9  LTD @(ASIS) @ADD1
10 LTD @(ASIS)@ADD2
11 LTD @(SP3 NEWPARA) DEAR @FIRSTNAME @LASTNAME:
12 LTD @(SP2) YOUR INTEREST IN VISITING OUR
13 LTD STATE IS AS IMPORTANT TO US AS IT IS TO YOU.
14 LTD @(SP2 NEWPARA) OPPORTUNITIES FOR TRAVEL AND TOURISM TO
15 LTD AND WITHIN "AMERICA'S" SUNSHINE STATE ARE GREAT BUT
16 LTD WE WANT THEM TO BE EVEN BETTER AND MORE AVAILABLE
17 LTD TO OUR VISITORS. INFORMATION ABOUT YOUR TRAVEL HELPS
18 LTD US TO PLAN AN EVEN MORE EXCITING STAY THE NEXT TIME
19 LTD YOU VISIT OUR STATE.
20 LTD @(SP2) VERY TRULY YOURS,
21 LTD @(SP4) FLORIDA STATE DEPARTMENT OF TOURISM
22 LTD @(SP3) ENCLOSURES: TRAVEL BROCHURES

23 ENDLTD

24 LETTER 10
```

Figure 707 Sample Letter Input Statements

The line numbers help explain the VISION:Results statements.

The FILE and field definition statements must precede the statements defining your letter. The LTH header (statement 6) is where you assign a unique application number and define the letter width and length, among other options described in detail in [Input to VISION:Results on page 729](#). The letter body follows, each text line preceded by the keyword LTD. The keyword ENDLTD (statement 23) must immediately follow the last detail text line.

The command LETTER, followed by an application number that ties it to the LTH statement, actually starts the printing of your letters (statement 24). Even though you can define a letter and specify all its detail text in your program, no letters are printed unless you include the LETTER command in your procedure logic.

The examples that follow show you how to generate various types of letters, including inserting variable data, using the From-To option, producing address labels, and using the laser option.

Input to VISION:Results

The following types of document definition statements are used:

LTH (header)	Only one LTH header can be submitted per document.
LTD (detail)	The actual text of the document is coded in the detail statements. You also enter instruction codes to determine line spacing of the document and other editing features with this statement.
ENDLTH (end letter)	This statement marks the end of a unique letter; it must follow the last LTD detail statement and must precede the LTH header defining another unique letter.

Your FILE and field definition statements (defining all of the variable fields used in the letter) must precede your letter definition statements.

LTH Statements and Keywords

Use the LTH header statement ([Figure 708](#)) to specify the general layout of your letter.

```
LTH nn [nnn WIDE] [nn LONG]
[NOEJECT] [LASERNn]
[INDENT nnn] [DYLEXTxx nnnK [WRITEALL]]
[IDLENGTH nn] [DYLTRxx [BY yyy] {DISK xxxx|TAPE}SYSxxxx]
[ASA] [TWOUP [xxx BETWEEN] [SWITCH] [LEFTSAVE nnnK]]
[OPTIMIZE | NOOPTIMIZE]
```

Figure 708 Syntax of LTH Statement

LTH	Required. Describes the application as letter writing and denotes this as a header statement. Always enter LTH as the first parameter on the header statement. Must be preceded by the FILE and field definition statements.
nn	Required. A unique application number must be assigned to each letter generated in the same program. This number must be an integer with a value from 1 to 99.
nnn WIDE	An integer value from 1 to 260. Specifies the number of characters per line (line width) in the printed letter. If omitted, this value defaults to 80. For any single application, the sum of the WIDE, INDENT, and IDLENGTH values must not exceed 132 characters (131 if LASERnn is specified). If DYLEXTxx with WRITEALL is not used, the maximum character length is 260. If the integer value is greater than 132 (131 if LASERnn is specified), DYLTRxx or DYLEXTxx must be specified. If DYLEXTxx is specified but DYLTRxx is not specified and nnn is greater than 132 (or 131), the WRITEALL option must be specified.
nn LONG	An integer value from 0 to 99. Specifies the number of lines per page requested for each letter. If zero, no line counter is maintained. 0 LONG plus the NOEJECT keyword are typically coded on the LTH statement when you want to produce address labels on continuous forms (gummed or Cheshire). If omitted, this value defaults to 56.
NOEJECT	Overrides automatic page ejection at the beginning of each letter. This feature is useful when using preprinted forms with pre-posted numbers.
LASERnn	Sets an indicator to determine the maximum number (n) of character sets per letter. A maximum number of 4 can be coded. If, for example, a value of 4 is specified, VISION:Results allows you to use four separate or unique character sets within your letter. If no value is specified, nn defaults to 4.
INDENT nnn	Specifies the number of spaces to indent the letter. If, for example, a value of 4 is specified following INDENT, every line of the entire letter is indented four spaces before printing. This option moves the letter to the right and better positions your letter on the output page. A numeric value between 1 and 100.

DYLEXTxx	The name of the subroutine that is loaded once and called each time a line is written to the output letter file (AUDPRINT/SYS009 or DYLTRxx), where xx is alphanumeric (AA-ZZ, 00-99). When the subroutine is called, Register 1 contains the address of a parameter list as follows: ■ Parameter 1 has the address of the print record. ■ Parameter 2 has the address of a work area defined as follows: – 1-byte flag (normally blank)—contains an E when this is the last time the EXIT is called. – 3-byte packed decimal field—contains the record length of the print record. ■ Parameter 3 has the address of DYLPARM, which is a 38-byte character field that can be manipulated from VISION:Results.
nnnK	Specifies the size of the subroutine in K, for VSE only.
WRITEALL	Specifies that the EXIT subroutine handles the writing of all the print records. No records are written to AUDPRINT/SYS009.
IDLENGTH nn	Contains the length of the identification field. If a value is specified, VISION:Results automatically generates the reserved word DYLCNxxID, where xx is the unique application number specified following LTH. The length of DYLCNxxID is dependent upon the length specified after IDLENGTH. A value of your choice can be moved to this reserved word to identify or sequence each letter. This value is placed in the rightmost position of each record line. A 132-byte fixed-length record is used. If an IDLENGTH of 5 were specified, the identification field value (DYLCNxxID) is placed at positions 128 to 132 of each line. This feature is useful when you are generating many different letters or labels that can be interspersed. A numeric value from 1 to 99.

DYLTRxx	The ddname or DLBL name of the output letter file to which each output line of the letter is written, where xx is alphanumeric (AA-ZZ, 00-99). If this keyword is not specified, each line of the letter is written to AUDPRINT/SYS009; in this case, the length of the print line must not exceed 132 characters. BY yyy This is the optional blocking factor for the output letter file defined by DYLTRxx. If not specified, a blocking factor of 10 is assumed. DISK xxxx VSE only. Required if DYLTRxx is used and the output letter file is to be written to disk. It specifies the type of disk for the output letter file. Valid entries for xxxx are 2311, 2314, 3330, 3340, 3350, 3375, 3380, FBA. TAPE VSE only. Required if DYLTRxx is used and the output letter file is to be written to tape. Specifies that the output letter file is to be written to tape. SYSxxx VSE only. Required if DYLTRxx is used. The SYS number for the output letter file. Valid entries for xxx are 000 to 240.
ASA	Specifies that ASA carriage control characters are to be used for the output letter file instead of the default (machine coded carriage control characters). This entry can be used only if the DYLTRxx keyword is used or when DYLEXTxx is used with the WRITEALL option. No ASA print records can be written to AUDPRINT/SYS009.
TWOUP	Use this to specify two-up letters (letters that are printed two to a page; the first letter on the left, the second on the right). xxx BETWEEN If used, TWOUP must be specified. Specifies the number of bytes to leave between the left-side letter and the right-side letter. The length of the print line is twice the width of the letter, plus the number of bytes between the letters, plus the length of the ID field, plus the length of the indent field on the LTH statement. xxx should be specified so that the length of the print line does not exceed 260 (132 if AUDPRINT/SYS009 is used or 131 if LASERnn is specified using AUDPRINT/SYS009). The default value for xxx is 5.

SWITCH	If used, TWOUP must be specified. Produces a 1-byte field DYLCNxxSW, where xx is the application number. This field can be set by you to control when (based on the logic for the letter) and where (left side or right side of the page) the next letter command is executed. If DYLCNxxSW is set to Y, the next letter command starts executing with reference to the other side (the left side if the current side was right or the right side if the current side was left) of the page. DYLCNxxSW is automatically reset to blank after each LETTER xx command is executed. If SWITCH is not used, the next letter command starts executing with reference to the other side of the letter, automatically, whenever a page eject occurs.
LEFTSAVE nnnK	If used, TWOUP must be specified. Specifies the amount of memory in K for holding the letter on the left side of the page. nnnK must be at least the width of the letter times the maximum number of lines before switching to the letter on the right side of the page. The default value for nnnK is 10K.
OPTIMIZE	OPTIMIZE on the LTH statement causes VISION:Results to attempt to write out as few lines as possible, yet achieve the requested printed letter. OPTIMIZE also causes space zero printing, @(SP0), to space zero lines and print the line.
NOOPTIMIZE	NOOPTIMIZE causes VISION:Results not to try to optimize the letter writing process. NOOPTIMIZE is the default.

For more information about OPTIMIZE, see [Optimize and Nooptimize on page 746](#).

LTD Statements and Keywords

The actual text of your letter immediately follows your LTH header statement. Each line of your text must begin with the LTD keyword (except when you are continuing a fixed print position line, as shown in [Figure 709](#)) that demonstrates the layout of the LTD text detail lines, edit commands, line spacing, and carriage control features available to you. The syntax of the LTD statement is shown in [Figure 710 on page 734](#)

```

FILE PRODFILE FB 352 5280
NAME          25   85
ADD1          25  110
ADD2          25  135
ACCOUNT       7    4
EXPDATE       6   44
PRODTYPE      12   50
LTH   01   80 WIDE   50 LONG   NOEJECT  INDENT 4
LTD1  @30 GEORGE & CO.
LTD   @26 17418 MAIN STREET
LTD   @22 LOS ANGELES, CALIF. 91344
LTD   @(SP3 ASIS)@NAME @53 @ACCOUNT(B)
LTD   @(ASIS)@ADD1

```

Figure 709 Letter Writing Application: Input Statements (Page 1 of 2)

```
LTD    @(ASIS)@ADD2
LTD    @(SP2 ASIS)DEAR CUSTOMER:
LTD2   @(SP2) THIS LETTER IS BEING SENT TO YOU TO ENABLE OUR CONTRACT
LTD    ADMINISTRATION DEPT. TO CONFIRM THE CORRECTNESS OF OUR RECORDS.
LTD    ON NOVEMBER 30, 1997 OUR RECORDS OF YOUR ACCOUNT #@ACCOUNT(Z)
LTD    SHOWED YOU ARE CURRENTLY LEASING @PRODTYPE, AND THAT YOUR
LTD    LEASE IS DUE TO EXPIRE ON @EXPDATE(D).
LTD3   @(SP2 NEWPARA) SINCE YOUR LEASE WILL EXPIRE BEFORE JULY 31, 1998,
LTD    YOU MAY AVOID OUR ANTICIPATED 10% PRICE INCREASE FOR FISCAL YEAR
LTD    1998/99 BY RENEWING NOW FOR A TWO-YEAR PERIOD. TO EXTEND YOUR
LTD    LEASE, SIMPLY SIGN BELOW AND RETURN THIS LETTER TO OUR OFFICES,
LTD4   @(SP2) @45 SINCERELY,
LTD    @(SP4) @45 GEORGE & CO.
LTD    @45 CONTRACT ADMINISTRATION
LTD    @(SP6 ASIS)YES, PLEASE RENEW OUR LEASE FOR A TWO-YEAR PERIOD.
LTD    @(SP4) @5 COMPANY NAME: _____
LTD    @(SP3) @5 BY: _____

ENDLTD

LETTER 01
STOP
```

Figure 709 Letter Writing Application: Input Statements (Page 2 of 2)

```
LTDnn [@(NEWPARA [Lnn|Inn]) | @(ASIS)]
      [@(SPn) | @(CHnn) | @(EJECT)]
      [nnn] [@(CSnn)]
      [dataname[editcode[,printsize]]]
ENDLTD
```

Figure 710 Syntax of LTD Statement

LTDnn Describes the application as letter writing and denotes this as a detail text statement. Always enter LTD as the first command on all text lines, except to continue a fixed print position line. For example:

```
LTD @10 LEASE START DATE @30 LEASE PAYMENT
    @70 PRODUCT TYPE @85 CUSTOMER NO.
```

@dataname All variable information to be inserted into your text must be prefixed with an @ (at) sign, followed by any valid data name.

To edit the numeric variable information you are inserting (such as suppressing leading zeroes and inserting commas), the data name must be immediately followed by an edit code, as shown in [Figure 709](#): @ACCOUNT(Z), @EXPDATE(D). Edit codes are described later in this section.

The default edit format is P; therefore, leading zeroes in numeric data are not automatically suppressed.

A print length can be specified for edited data names. For example, @ACCOUNT(Z,10) means that the data in the field ACCOUNT is printed in Z format within 10 print positions and is right-aligned in those 10 positions. If necessary, high-order digits are truncated. The length option cannot be used for non-numeric, non-edited fields. For more information about edited data names, see [Example 4](#) on page [754](#).

Do not use the print length specification in conjunction with edit code G, J, K, or L because it can truncate the leading minus sign and cause alignment problems.

You can also place a character in front of @dataname. For example, #@ACCOUNT. The character placed in front of @dataname immediately precedes your variable information with no intervening spaces (for example, #203345).

@(SPn)

The line spacing keyword, enclosed within parentheses, must be preceded by the @ (at) sign. SP is followed by a numeric value from 0 to 9; SP0 allows for zero line spacing and is used for boldface printing on impact printers. Specifies to space n number of lines, up to a maximum of nine, before printing the line on which this keyword is specified. In [Figure 709](#), double-spacing occurs before the line containing DEAR CUSTOMER: is printed.

If you want more line spacing than can be performed with the nine maximum allowed, code as many @(SPn) lines as necessary. For example:

```
LTD @(SP9)
LTD @(SP4)
```

The example above generates 13 blank lines. The (SPn) keyword need not be followed by text.

Automatic editing is assumed for any text preceded by @(SP1) unless @(SP1 ASIS) is coded. @(NEWPARA) is assumed for any text preceded by @(SP2) through @(SP9), inclusive.

Note: The (SPn), (CHnn), and (EJECT) keywords are mutually exclusive: if you use more than one of them together, the last one coded is the option performed.

@(CHnn) Specifies channel control spacing. CH1–CH12 handle channels 1–12, respectively. @(CH1) ejects the page and is identical to @(EJECT).

Use the COPY DYCHANNEL statement when performing channel skipping with two-up processing. For more information, see [Channel Skipping with Two-up Letter Processing on page 745](#). If the COPY DYCHANNEL statement is missing, error message DYL-807E DATA NAME OF DY01CHANNEL NOT DEFINED may occur.

Note: The (SPn), (CHnn), and (EJECT) keywords are mutually exclusive: if you use more than one of them together, the last one coded is the option performed.

@(EJECT) Specifies page ejection to occur prior to printing the current or next line. Identical to @(CH1).

Note: The (SPn), (CHnn), and (EJECT) keywords are mutually exclusive: if you use more than one of them together, the last one coded is the option performed.

@(CSnn) Specifies the character set that is used for the text that follows this keyword. This allows you to change character sets within the same line of text. For example:

```
LTD @(CS1) DANGER @(CS2) @25 HIGH EXPLOSIVES @(CS1) DANGER
```

A number from 1 to 4 can be coded for n; it represents the character sets defined in your JCL. See [Laser JCL on page 743](#).

If you use this command, you must use LASERnn in the LTH statement. Do not exceed the LTH LASERnn limit.

@(ASIS) Specifies that no editing (removing unnecessary blanks) is done to the text that follows this keyword (that is, the text is to be left as is). This is useful if you want to print columns across your letter page. You can precede the text with the @(ASIS) keyword and leave spaces between the columnar material, as follows:

```
LTD @(ASIS) 1996          1997          1998
```

1996 starts in print position 1 because there are no spaces preceding it. Eight spaces follow 1996. 1997 and 1998 print, also separated by eight spaces. If the @(ASIS) were omitted here (that is, no edit command was specified for this text line), automatic text compression is assumed: 1996 starts in print position 1, and 1997 and 1998 follow, each separated by only one space.

@nnn

The @ (at) sign must precede the numeric value you specify when you want the text following it to be printed in that assigned print location. In [Figure 709](#), 17418 MAIN STREET is printed starting at position 26. When an @nnn is placed in any line, that line automatically becomes an ASIS line. For example, you can specify the following:

```
LTD @5 1996 @15 1997 @25 1998
```

This example prints 1996 at position 5, 1997 at position 15, and 1998 at position 25.

If @nnn is immediately followed by @dataname, the first non-blank character in the data name prints at position nnn.

If you specify text to print at a location that exceeds your line width value (characters per line WIDE), an error message is generated.

@(NEWPARA
[Lnn | Inn])

@(NEWPARA) specifies that a new paragraph is wanted, so the preceding text line is left unfilled if it contains fewer characters than the number WIDE specified in your header statement. Double-spacing does not automatically occur before printing the new paragraph; you must specify, for example, @(SP2 NEWPARA) if you want a blank line before the first line of the new paragraph. All text on the statement line following @(NEWPARA) is compressed (that is, automatic word wrap-around and removal of extraneous blanks occurs).

The (SPn), (CHnn), and (EJECT) keywords are mutually exclusive: if you use more than one of them together, the last one coded is the option performed.

Lnn specifies that a paragraph starts with the first word of the paragraph indented nn spaces. Inn specifies that all the lines in the paragraph are indented nn spaces. nn cannot exceed 20.

ENDLTD

Enter ENDLTD as the final command after the last detail text line (LTD statement). It must precede an LTH header defining another unique letter. An error message is generated if you do not include this statement.

Additional LTD Statement Information

Use the following additional guidelines when entering LTD statements in VISION:Results:

- If you do not specify @(SPn), @(CHnn), or @(EJECT) on a detail text line (LTD statement), the default is single-spacing. If you do not specify @(ASIS) or @nnn on a detail text line, the default is automatic text compression. All words are separated by exactly one blank, and as many words as will fit into the maximum line width (the WIDE value specified in your LTH statement) are placed into the printed line.
- Never break or hyphenate words at the end of your LTD statements. If you do so, each of the two word parts is treated as a full word and the two parts are printed separated by a single space. VISION:Results does not support word hyphenation.
- You can specify more than one text edit option keyword following your LTD command. They can be in any order, but they must be coded together within one set of parentheses (except @nnn, which cannot be enclosed in parentheses) and separated by at least one space. For example:

Valid: @(SP2 NEWPARA)

Invalid: @(SP2) (NEWPARA)

- Unless you precede your text with an @nnn, all text begins at position 1 of the print line.
- To enclose text in double quotation marks in your letter, you must enclose that text in single quotation marks (in addition to the double quotation marks) on the LTD statement.

For example, LTD "'Restaurants of Southern California'" prints as "Restaurants of Southern California" in your letter.

Conversely, if you want text to be enclosed in single quotation marks in your letter, you must enclose that text in double quotation marks (in addition to the single quotation marks) on the LTD statement.

For example: LTD "'Restaurants of Southern California'" prints as 'Restaurants of Southern California' in your letters.

- All edit, line spacing, and carriage control keywords must be specified at the beginning of your text line, with the exception of the @nnn and variable information (@ followed by a data name), both of which can be coded anywhere.
- To use the @ sign or any character string that also is a command (LTH, LTD, ENDLTD) or keyword within the letter body, enclose them within either single or double quotation marks (for example, 'LTD' or "LTD").
- Program FILE and field definition statements must be specified before letter definition statements.
- In program procedure logic, the LETTER nn command invokes the printing of letters. If you omit this invoking command, no letters print, even though one may have been fully defined. For more information, see [LETTER Command on page 742](#).

LTD Edit Codes

The valid edit codes and their values are described in the following table.

Edit Code	Description	Examples	
E	Edit with zero suppression, commas, decimal insertion, and negative sign to the right of the field. Blank if zero.	001234.56 prints as 000000.00 prints as	1,234.56 (blank)
Y	Leading -. Edit with zero suppression, commas, decimal insertion, and negative sign to the left of the number. Blank if zero.	-00123456 prints as 000000.00 prints as	-1,234.56 (blank)
F	Leading \$. Edit with zero suppression, commas, decimal insertion, and negative sign to the right of the field. Blank if zero. Float a currency sign to the left of the field.	-00123456 prints as 000000.00 prints as	\$1,234.56- (blank)
G	Leading -\$. Edit with zero suppression, commas, decimal insertion, and negative sign to the left of the field. Float a currency sign to the direct left of the field. Blank if zero.	00123456 prints as 000000.00 prints as	-\$1,234.56 (blank)
K	Leading \$-. Edit with zero suppression, commas, decimal insertion, and negative sign to the direct left of the field. Float a currency sign to the left of the negative sign. Blank if zero.	00123456 prints as 00000000 prints as	\$-1,234.56 (blank)
A	Edit with zero suppression, commas, decimal insertion, and negative sign to the right of the field. Print decimal point and zeros to the right of decimal. If field has no decimal positions and has a zero value, nothing prints.	001234.56 prints as 000000.00 prints as 000000000000 prints as	1,234.56 .00 (blank)

Edit Code	Description	Examples	
X	Leading -. Edit with zero suppression, commas, decimal insertion, and negative sign to the left of the field. Print decimal point and zeros to the right of decimal. If field has no decimal positions and has a zero value, nothing prints.	001234.56 prints as	1,234.56
		-001234.56 prints as	-1,234.56
		0000000.00 prints as	.00
		00000000000 prints as	(blank)
H	Leading \$. Edit with zero suppression, commas, decimal insertion, and negative sign to the right of the field. Float a currency sign (the default is \$) to the left of the most significant digit. Blank if zero.	001234.56 prints as	\$1,234.56
		-001234.56 prints as	\$1,234.56-
		0000000.00 prints as	\$.00
		00000000000 prints as	(blank)
J	Leading -\$. Edit with zero suppression, commas, decimal insertion, and negative sign to the left of the field. Float a currency sign to the direct left of the field. Blank if zero.	001234.56 prints as	\$1,234.56
		-001234.56 prints as	-\$1,234.56
		0000000.00 prints as	\$.00
		00000000000 prints as	(blank)
L	Leading \$. Edit with zero suppression, commas, decimal insertion, and negative sign to the direct left of the field. Float a currency sign to the left of the negative sign. Blank if zero.	001234.56 prints as	\$1,234.56
		-001234.56 prints as	\$-1,234.56
		0000000.00 prints as	\$.00
		00000000000 prints as	(blank)
Z	Edit with zero suppression, decimal insertion, negative sign to the right of the field, but no commas. Blank if zero.	001234.56 prints as	1234.56
		000000.00 prints as	(blank)
U	Leading -. Edit with zero suppression, decimal insertion, negative sign to the left of the field, but no commas. Blank if zero.	00123456 prints as	1234.56
		-00123456 prints as	-1234.56
		000000.00 prints as	(blank)

Edit Code	Description	Examples	
B	Edit with zero suppression, commas, decimal insertion, and negative sign to the right of the field. Blank if zero. If zero value, print decimal point and zeros to the right of the decimal. If field has no decimal positions and has a zero value, nothing prints.	001234.56 prints as	1,234.56
		000000.00 prints as	(blank)
Q	Leading -. Edit with zero suppression, decimal insertion, negative sign to the left of the field, but no commas. Blank if zero. If zero value, print decimal point and zeros to the right of the decimal. If field has no decimal positions and has a zero value, nothing prints.	001234.56 prints as	1234.56
		-001234.56 prints as	-1234.56
		000000.00 prints as	.00
		000000000 prints as	(blank)
P	Default edit code. Edit with decimal insertion, negative sign to the right of the field, but no commas. If the field has no decimal positions and has a zero value, zeros print. If it has a zero value, zeros print.	001234.56 prints as	001234.56
		000000000 prints as	000000000
W	Leading -. Edit with decimal insertion, negative sign to the left of the field, but no commas. If the field has no decimal positions and has a zero value, zeros print. If it has a zero value, zeros print.	-001234.56 prints as	-001234.56
		0000000.00 prints as	0000000.00
		000000000 prints as	000000000
D	Date edit of a 6-byte zoned decimal or 4-byte packed field.	011599 prints as	01/15/99

Edit Code	Description	Examples	
V	Edit a date field containing a 4-digit year. The date field must be either an 8-byte zoned decimal or 5-byte packed field.	01151999 prints as	01/15/1999
S	Edit a social security number from a 9-byte numeric or 5-byte packed field.	999999999 prints as	999-99-9999

See [Edit Codes on page 313](#) for a list of all VISION:Results edit codes.

LETTER Command

The LETTER command, followed by the application number, starts the VISION:Results letter writing function in your procedure logic. The unique application number ties this command to your LTH header statement and uses the criteria specified in that statement (such as page width and length) to format your letter.

```
VISION:Results FILE statement
and field definition statements
:
:
LTH 01 75 WIDE 66 LONG
LTD
. (Body of letter)
:
ENDLTD
LETTER 01
```

Figure 711 LETTER Command

This LTH statement specifies a letter length of 66 lines and width of 75 characters. No IDLENGTH or INDENT has been specified; it defaults to a page eject before each letter.

The LETTER command starts the printing of these letters. Without this invoking command, no letters are printed, even though a letter has been fully defined in your program.

The LETTER command and application number can be followed by From-To line numbers to print a range of letter lines. For a description of this option, see [Example 5 on page 755](#).

Laser JCL

Note: This section applies to z/OS only.

When using the laser option, an additional JCL DD statement is required. An example follows with an explanation of the DD parameters:

```
//AUDPRINT DD SYSOUT=(1,,DYL1),FCB=DYL1,
//      CHARS=(GP12,SB12),BURST=N,
//      DCB=(BLKSIZE=80,RECFM=FA,OPTCD=J)
```

Figure 712 Laser JCL (z/OS) Example

The ddname AUDPRINT is required and must be coded this way.

The DD parameters are:

SYSOUT=(class,program,form)

The SYSOUT= parameter routes the print to the system output device. This is determined by installation standards.

Class—output class.

Program—name of the program in SYS1.LINKLIB.

Form—request to the operator to mount special forms.

FCB=image-id

The FCB parameter specifies the forms control image for printers and controls the movement of forms on a printer.

CHARS=(table, ..., table)

Specifies which character sets are available in this run. When multiple character sets are specified, the OPTCD=J parameter must be coded.

BURST=Y or N

This parameter specifies whether output should be burst into separate sheets (Y) or in continuous fanfold mode (N).

DCB=(..., OPTCD=J)

When multiple character sets are specified, this parameter specifies the byte that selects the table used to print a line.

Letter Writing Considerations

Take these points into consideration when using the letter writing function:

- If the letter is using a data name in the buffer area of a VSAM file and the letter command is started after end of file, the job aborts with a system 731 abend message. To avoid this, move either the fields or the entire record in the buffer to a work area. The data names used are defined by your work area, not your buffer area.
- No code should follow the keyword ENDLTD. This command must stand alone on the statement line.
- If From and To are used with the LETTER invoking command and there are no corresponding lines in the letter definition to match this range, no letter is generated.
- If data within a variable extends beyond a line's width in an @(ASIS) line, the variable data is continued on the next line of print, forcing an extra line into the letter.
- Literals (including phrases enclosed within quotation marks) and data names cannot be continued onto a second line.
- If any letter keywords or commands are included in the text of a letter, they must be enclosed within single or double quotation marks.
- The @ sign cannot be used as a constant in the text of a letter. If @ is required in your print line, it must be contained in a field and the data name of the field must be referenced in your letter text.
- If an apostrophe is used in a string, that string must be in quotation marks. For example:

To produce the output:	IS THIS MY BROTHER'S SHOE?
Specify:	IS THIS MY "BROTHER'S" SHOE?
To produce the output:	'IS THIS MY BROTHER'S SHOE?'
Specify:	"'IS THIS MY BROTHER'S SHOE?'"

- If a packed decimal field is printed without zero suppression and it is zero in value, it is printed with the first position blank. This blank position is the significant start character in the edit mask.
- If an LTD statement has a quotation-marked data string and there are commas in the statement, place the comma inside the outer quotation marks. For example:

```
THE FOUR BOOKS "'EXODUS'," "'MILA-18'," "'TRINITY'" AND "'QB-VII'".
```

not:

```
THE FOUR BOOKS "'EXODUS'", "'MILA-18'", "'TRINITY'" AND "'QB-VII'".
```


- If you are using @nnn to specify exact positioning of your text, any blanks between the @nnn and the first non-blank character are ignored. In the example:

```
LTD @10      1996      1997 @27      1998
```

1996 starts at position 10 and 1998 starts at position 27 — the spaces between @10 and 1996, and @27 and 1998, are dropped. However, all other extraneous blanks in the line (for example, between 1996 and 1997) are not removed.

- If you specify @(ASIS) on your LTD statement, blanks following @(ASIS) and the first non-blank character are not removed. In the example:

```
LTD @(ASIS)      1996      1997
```

1996 and 1997 are preceded by five blanks.

- Letters cannot be copied into a VISION:Results program using the COPY, COPYP, COPYL, or COPYC command.

Channel Skipping with Two-up Letter Processing

When channel skipping with two-up letter processing, you must specify to VISION:Results where the channels are physically on the printed page. The COPY member DYCHANNEL is provided to help with the channel specification. You must specify the application number and the line number, relative to the top of the page, associated with each channel. The format of the COPY DYCHANNEL statement is as follows, where #N= is the application number of the associated letter (00 to 99) and #1= through #C= are the line numbers for channels 1–12, respectively:

```
COPY DYCHANNEL #N= #1= #2= #3= #4= #5= #6= #7=
                  #8= #9= #A= #B= #C=
```

Only used channels need to be specified. The line numbers must be progressively larger as the channels increase. For example, the line number for channel 6 must be greater than the line number for channel 5. If channel 1 is not specified, it defaults to 1, the first line on the page.

The COPY DYCHANNEL statement must be coded before the LTH statement for the associated letter. For example, to use channel skips 2, 4, and 7 in a two-up letter where channel 2 is line 5, channel 4 is line 15, and channel 7 is line 32 on the page, you code:

```
COPY DYCHANNEL #N=01 #2=5 #4=15 #7=32
```

```
LTH 1 TWOUP
LTD @(CH2) @1 THIS IS THE FIRST LINE.
LTD @(CH4) @1 THIS IS THE SECOND LINE.
LTD @(CH7) @1 THIS IS THE THIRD LINE.
ENDLTD
```

Optimize and Nooptimize

Placing the keyword OPTIMIZE on the LTH statement writes as few lines as possible, yet achieves the requested printed letter. NOOPTIMIZE disables the optimize feature during the letter writing process. NOOPTIMIZE is the default.

When channel skipping is performed with OPTIMIZE, the line numbers of the channels to be used must be specified using the COPY DYCHANNEL statement. For a description of this procedure, see [Channel Skipping with Two-up Letter Processing on page 745](#). VISION:Results channel skips when this results in fewer lines being generated. DYCHANNEL is activated only if the letter detail contains at least one @(CHnn) command.

OPTIMIZE also causes space zero printing to work differently. Normally, @(SP0) means print the line and space zero lines, while @(SPn) (where n is greater than zero) means space n lines and print the line. The OPTIMIZE keyword makes @(SP0) consistent with @(SPn); the spacing operation becomes, in all cases, space before and write the print line.

The following example demonstrates the difference between using OPTIMIZE and not using OPTIMIZE.

```
COPY DYCHANNEL #N=01 #1=1 #3=4 #4=10 #7=20 #8=40

LTH 1 ASA DYLTRAB
LTD @(EJECT) @1 THIS IS LINE 1 - CHANNEL 1.
LTD @(SP3)   @1 THIS IS LINE 4 - CHANNEL 3.
LTD @(SP7)   @1 THIS IS LINE 11.
LTD @(CH8)   @1 THIS IS LINE 40 - CHANNEL 8.
ENDLTD
```

The above code generates the following nine lines for each letter.

Code	Line Number
1THIS IS LINE 1 - CHANNEL 1.	1
	2
0THIS IS LINE 4 - CHANNEL 3.	3
	4
-	5
-THIS IS LINE 11.	6
	7
8THIS IS LINE 40 - CHANNEL 8.	8
	9

The first position of each line is the ASA carriage control character.

The following example demonstrates the effect of placing the keyword OPTIMIZE on the LTH statement:

```
LTH 1 ASA DYLTRAB OPTIMIZE
```

The above code generates the following five lines for each letter.

Code	Line Number
1THIS IS LINE 1 - CHANNEL 1.	1
-THIS IS LINE 4 - CHANNEL 3.	2
4	3
THIS IS LINE 11.	4
8THIS IS LINE 40 - CHANNEL 8.	5

Without specifying OPTIMIZE, nine lines per letter are generated; while, with OPTIMIZE, only five lines per letter are generated. The printed letter is identical in either case, assuming that channel 4 is line 10 of the page.

Letter Writing Examples

The following sections are examples to help you with the letter writing features.

Example 1 Specifying a Letter

George & Co., a wholesale distributor, wants to send a letter to each of its customers announcing a price increase on merchandise. The company's input file, stored on tape, contains numerous records, one for each account. The fields defined in the following table illustrate some of the information stored concerning each customer. The information on this tape provides the variable data needed for insertion in each letter.

File characteristics: format = blocked, record size = 352, block size = 5280

Field Size	Field Position	Description	Data Type	Decimal	Data Name Used
7	4	ACCOUNT NUMBER	NU		ACCTNO
6	44	BILLING DATE	CH		BILLNGDATE
25	85	ACCOUNT NAME	CH		NAME
25	110	STREET ADDRESS	CH		ADD1
25	135	CITY,STATE,ZIP	CH		ADD2

5	170	BALANCE AMOUNT	PD	2	BALANCE
2	182	ACCOUNT CODE	CH		ACCTCODE

Key: NU = numeric, CH = character, PD = packed decimal

Example 1— Input to VISION:Results

[Figure 713](#) shows the letter writing statements and VISION:Results program required to generate the letter described above.

```
1  FILE ARFILE INPUT FB 352 5280
2  NAME 25 85
3  ADD1 25 110
4  ADD2 25 135
5  ACCTNO 7 4 NU
6  LTH 01 60 WIDE 30 LONG
7  LTD @20 GEORGE & CO.
8  LTD @20 16255 VENTURA BLVD.
9  LTD @20 ENCINO, CALIFORNIA
10 LTD @(SP2) @42 @DYLDATE
11 LTD @(SP2 ASIS)@NAME
12 LTD @(ASIS)@ADD1
13 LTD @(ASIS)@ADD2
14 LTD @(SP2 ASIS)DEAR CUSTOMER:
15 LTD @(SP2) ON DECEMBER 6, 1998, OUR PRICES MUST BE RAISED 15% ON
16 LTD STEREO EQUIPMENT, TAPE DECKS AND OTHER ITEMS. WE URGE YOU
17 LTD TO PLACE HOLIDAY SEASON ORDERS NOW TO AVOID THESE INCREASES.
18 LTD @(SP2) @42 YOURS TRULY,
19 LTD @(SP4) @42 SALES MANAGER.
20 ENDLTD
21 LETTER 01
```

Figure 713 [Example 1](#) Letter Writing Input Statements

The numbers at the left of these statements are used to discuss the various features of the letter and how they are invoked.

Statement 1 is the FILE statement that defines the input file, ARFILE, containing the variable information to insert in each letter.

Statements 2–5 are the field definition statements, where data names are assigned to the fields that contain the variable information needed in the letters.

In statement 6 (the LTH header), a 60-character line width and 30-line page length is specified. VISION:Results defaults to a page eject at the beginning of each letter. No INDENT or IDLENGTH values have been specified.

LTD statements 7–9 specify that the company name, street address, city, and state are to be printed starting at position 20. No line spacing has been specified, so these three lines are single-spaced.

Following the company name and address, LTD statement 10 specifies that double-spacing is to occur; the current date is printed at position 42. The prefix @ precedes the reserved word DYLDATE (the current date).

LTD statement 11 specifies that double-spacing is to occur after the current date, and the variable information contained in the field NAME is printed starting in line position 1. On the next two lines, the variable information in the fields ADD1 and ADD2 is printed, each beginning at position 1. @(ASIS) is specified on each of the address lines so that text compression does not occur. If @(ASIS) were omitted here, ADD2 would follow ADD1 on the same line.

After double-spacing, the letter salutation DEAR CUSTOMER: (LTD statement 14) is printed. Statements 15–17 contain the body of the letter. Note that statements 16 and 17 contain no edit options; therefore, automatic text compression of the letter body occurs. The text wraps around within the 60-character print line specified in the header. The ENDLTD command terminates the letter statements.

Statement 21 is the procedure logic. The command LETTER 01 means that processing of a letter is to take place on each record in the input file. After processing, each letter is to be printed. The 01 application number that appears in the header statement and the LETTER command links these statements together. If two unique letters were coded in this program (the second defined by the application number 02), the LETTER 01 command would invoke processing of letter 01 only. If the LETTER command is not present, printing of letters is never invoked.

Example 1—VISION:Results Output

One of the letters generated by this program is shown in [Figure 714](#).

```
GEORGE & CO.  
16255 VENTURA BLVD.  
ENCINO, CALIFORNIA  
  
11/02/01  
  
ANDREWS, CHARLES  
1310 LOMA DR.  
CAMBRIA PINES, CA  
  
DEAR CUSTOMER:  
  
ON DECEMBER 6, 1999, OUR PRICES MUST BE RAISED 15% ON STEREO  
EQUIPMENT, TAPE DECKS, AND OTHER ITEMS. WE URGE YOU TO  
PLACE HOLIDAY SEASON ORDERS NOW TO AVOID THESE INCREASES.  
  
YOURS TRULY,  
  
SALES MANAGER.
```

Figure 714 [Example 1](#) Output—Letter Writing Generated Letter

Example 2 Selecting Records and Generating Reports

George & Co.'s marketing department has prepared a catalog of special items that they want to make available to their best customers. These customers all have an account code of MO on the accounts receivable file. The marketing manager wants a cover letter sent to all MO accounts, offering the catalog and a copy of the guide Restaurants of Southern California to all who respond by August 15th.

This example differs from [Specifying a Letter on page 747](#) in that record selection is being performed using procedure logic, so that only MO accounts receive letters, rather than all accounts on the file. In addition, a report is generated, listing accounts eligible to receive the special catalog.

Example 2—Input to VISION:Results

The required letter writing statements and VISION:Results program appear in [Figure 715](#).

```
1 COPY DYAUDREC
2 LTH 01 66 WIDE
3 LTD @23 GEORGE & CO.
4 LTD @23 16255 VENTURA BLVD.
5 LTD @23 ENCINO, CALIFORNIA
6 LTD @(SP2) @50 @DYLDATE
7 LTD @(SP2)@NAME @50 @ACCTNO
8 LTD @(ASIS)@ADD1
9 LTD @(ASIS)@ADD2
10 LTD @(SP2 ASIS)DEAR CUSTOMER:
11 LTD @(SP2) AS A VALUED CUSTOMER, YOU CAN RECEIVE OUR SPECIAL PURCHASE
12 LTD CATALOG IF YOU RESPOND BY AUGUST 15. JUST RETURN THE FORM BELOW
13 LTD AND THE CATALOG WILL BE ON ITS WAY, TOGETHER WITH A FREE COPY OF
14 LTD "RESTAURANTS OF SOUTHERN CALIFORNIA."
15 LTD @(SP2) @50 SINCERELY,
16 LTD @(SP4) @50 GEORGE & CO.
17 LTD @(SP2) @23 -----
18 LTD @(SP2 ASIS)PLEASE SEND MY CATALOG AND BONUS GIFT TO:
19 LTD @(SP2 ASIS)NAME: _____
20 LTD @(SP2 ASIS)ADDRESS: _____
21 LTD @(SP2 ASIS)CITY: _____ STATE: _____ ZIP: _____
22 ENDLTD
23 IF ACCTCODE EQ 'MO'
24     LETTER 01
25     LIST ACCTNO (ACCOUNT'NUMBER) NAME ADD1 (STREET) ADD2 (CITY'STATE)
26     BALANCE
27     ELSE REJECT ENDIF
28 ON FINAL
29     LIST SUM BALANCE WITH 2 BEFORE
30 T1 'CUSTOMERS ELIGIBLE FOR SPECIAL CATALOG' WITH 2 AFTER
```

Figure 715 Example 2 Letter Writing Input Statements

The numbers at the left are for reference purposes only.

In statements 18–21, note the use of the edit options @(SP2 ASIS), indicating to double-space between each of these lines and to omit automatic text compression.

The IF statement coded at lines 23–27 selects only those accounts with the MO code to receive the letter. The printing of the letters is invoked by the LETTER 01 command, statement 24. The LIST statement (statement 25) generates a report of all accounts receiving the letter.

Example 2—VISION:Results Output

[Figure 716](#) shows one of the letters generated, and [Figure 717](#) shows the report of customers eligible for the special catalog.

GEORGE & CO.
16255 VENTURA BLVD.
ENCINO, CALIFORNIA

08/01/01

MORENO, ELIZABETH
208 1/2 INEZ ST
NORWALK CA 90605

6039804

DEAR CUSTOMER:

AS A VALUED CUSTOMER, YOU CAN RECEIVE OUR SPECIAL PURCHASE CATALOG IF YOU RESPOND BY AUGUST 15. JUST RETURN THE FORM BELOW AND THE CATALOG WILL BE ON ITS WAY, TOGETHER WITH A FREE COPY OF "RESTAURANTS OF SOUTHERN CALIFORNIA."

SINCERELY,

GEORGE & CO.

PLEASE SEND MY CATALOG AND BONUS GIFT TO:

NAME: _____

ADDRESS: _____

CITY: _____ STATE: _____ ZIP: _____

Figure 716 [Example 2](#) Letter Writing Generated Letter

CUSTOMERS ELIGIBLE FOR SPECIAL CATALOG

ACCOUNT NUMBER	NAME	STREET	CITY STATE	BALANCE
6039804	MORENO, ELIZABETH	208 1/2 INEZ ST	NORWALK CA 90605	.00
6007724	HARRINGTON, RUTH	222 MAPLE AVE	LOS ANGELES	37.60
6017479	BERSON, ANNA	4159 FANCHO AVE SP9	LOS ANGELES CA	3.00
6012132	ZENZOLA, MICHAEL	9729 MOTOR AVE	BELLFLOWER CA 90706	30.20
2002922	FLOWERS, ETHEL	4420 ARKANSAS	LOS ANGELES CA 90048	2,541.70
6204384	FREDRICK, EMERY	573 VAN HORNE AVE	LOS ANGELES CA	15.00
6001327	CARLON, MR. MARIANO	3417 CEDAR	LOS ANGELES CA	118.00
8002053	JENNINGS, WILL	9560 N HAYWORTH 210	LOS ANGELES CA	56.35
				2,801.84

Figure 717 [Example 2](#) Letter Writing Report of Customers Eligible

Example 3 Generating Address Labels

George & Co.'s sales department has some special literature about upcoming merchandise. They want address labels generated for all records about the accounts receivable file and plan to use these labels for their mailing.

Example 3—Input to VISION:Results

[Figure 718](#) is the letter writing and VISION:Results program statements needed to generate the address labels.

```
1  FILE ARFILE INPUT FB 352 5280
2  NAME 25 85 CH ADD1 25 110 CH
3  ADD2 25 135 CH NAMEADDR 75 85
4  WORKAREA
5  NAMEADDR1 75 VALUE ' ' REINIT NAME1 25 1 ADD1A 25 26 ADD1B 25 51
6  NAMEADDR2 75 VALUE ' ' REINIT NAME2 25 76 ADD2A 25 101 ADD2B 25 126
7  NAMEADDR3 75 VALUE ' ' REINIT NAME3 25 151 ADD3A 25 176 ADD3B 25 201
8  FLAG 1
9  LTH 01 100 WIDE 0 LONG NOEJECT
10 LTD @1 @NAME1 @29 @NAME2 @57 @NAME3
11 LTD @1 @ADD1A @29 @ADD2A @57 @ADD3A
12 LTD @1 @ADD1B @29 @ADD2B @57 @ADD3B
13 LTD @(SP1)
14 ENDLTD
15 RD1:
16 READ ARFILE
17 FLAG = 1
18 MOVE NAMEADDR TO NAMEADDR1
19 RD2:
20 READ ARFILE
21 MOVE NAMEADDR TO NAMEADDR2
22 RD3:
23 READ ARFILE
24 FLAG = 0
25 MOVE NAMEADDR TO NAMEADDR3
26 LETTER 01
27 ACCEPT
28 ON END OF INPUT
29 IF FLAG NE 0
30 LETTER 01
31 ENDIF
32 STOP
```

Figure 718 [Example 3](#) Address Label Statements

The numbers at the left are for reference purposes only.

Statement 1 is the FILE statement that identifies the input file to be used and defines its attributes.

Statements 2 and 3 are field definitions (data names) of the name and address fields in the input file.

Statements 4–8 are field definitions of WORKAREAs used by the program during processing.

Statements 9–14 are the letter writing statements required to generate three-up labels for all records concerning the accounts receivable file. In the LTH statement, the lines per page has been set to zero and the NOEJECT keyword is coded so that the default page eject at 56 lines is not taken. (Continuous forms such as labels generally use 0 LONG.)

Statements 15–32 make up the procedure logic. Statements 15–27 read three records and move the address information for each record to the WORKAREA (NAMEADDR1, NAMEADDR2, NAMEADDR3). When invoked, the LETTER 01 command (statement 26) generates three-up labels, each consisting of three lines (the name and two address lines). A flag is set to 1 when the information to be printed in the leftmost label is read; a flag is set to 0 when the information to be printed in the rightmost label is read.

ON END OF INPUT logic is coded in statements 28–32. The LETTER 01 command is invoked if the FLAG does not equal 0. This ensures that every record on the file has an address label printed for it. Without this logic, printing of the last one or two labels would be skipped if the total number of records on the file was not divisible by three. Program processing is terminated.

Example 3—VISION:Results Output

[Figure 719](#) shows three of the address labels produced by this letter writing program.

TORRES, ERNESTO	CHO PYUNG, SUH	S. F. MEM. HOSP.
23444 PARK LANE	33333 PALL MALL	6789 OLD MAN RD
LOS ANGELES CA	GLENDALE CA 91206	L.A. CA 90023

Figure 719 [Example 3](#) Letter Writing Address Labels

Example 4 Specifying a Print Length for Edited Data Names

Previously in this chapter, there were demonstrations about how to specify a print length for edited data names, such as @AMOUNT(E,10), where the contents of the field AMOUNT are printed in E format, right-aligned, within 10 print positions. The print size is especially useful when printing numeric data in columns. For example, the program shown in [Figure 720](#) demonstrates the printing of different size numeric fields in a column. Note that in statements 25–29, each numeric field is to begin printing at position 45. By specifying the same print length (statement 14) for each data name, the decimals align when the five fields are printed. [Figure 721](#) shows the results of this program.

```

1  FILE ARFILE  INPUT  FB 352 5280
2  ACCTNO      7      4          ; ACCOUNT NUMBER
3  NAME        25     85
4  ADD1        25    110          ; STREET
5  ADD2        25    135          ; STATE ZIP
6  INSTLBAL    6    191 PD 2 A    ; INSTALLMENT BALANCE
7  INSTPAY     5    197 PD 2 A    ; INSTALLMENT PAYMENT
8  BALPART     4    202 PD 2 A    ; BALANCE PARTIAL PAYMENT
9  INTPART     3    206 PD 2 A    ; INTEREST PARTIAL PAYMENT
10 NUMPY       2    209 PD      A  ; NUMBER PAYMENT
11 LTH 01 66 WIDE
12 LTD @(SP2)
13 LTD @23 GEORGE & CO.
14 LTD @23 16255 VENTURA BLVD.
15 LTD @23 ENCINO, CALIFORNIA
16 LTD @(SP2) @50 @DYLDATE
17 LTD @(SP2)@NAME @50 @ACCTNO
18 LTD @(ASIS)@ADD1
19 LTD @(ASIS)@ADD2
20 LTD @(SP2 ASIS)DEAR CUSTOMER:
21 LTD @(SP2) YOU NOW HAVE THE OPTION OF PAYING YOUR BALANCE IN EASY
22 LTD MONTHLY INSTALLMENTS BY USING OUR REVOLVING CREDIT PLAN. IF
23 LTD YOU WISH TO DO SO PLEASE SIGN BELOW AND RETURN IN THE ENCLOSED
24 LTD ENVELOPE.
25 LTD @(SP2) @11 BALANCE @45 @INSTLBAL(A,14)
26 LTD @11 NO. OF MONTHLY INSTALLMENTS @45 @NUMPY(A,14)
27 LTD @11 MONTHLY PAYMENT @45 @INSTPAY(A,14)
28 LTD @11 PORTION APPLIED TO BALANCE @45 @BALPART(A,14)
29 LTD @11 INTEREST @45 @INTPART(A,14)
30 LTD @(SP2) @50 SINCERELY,
31 LTD @(SP4) @50 GEORGE & CO.
32 LTD @(SP4) @25 -----
33 LTD @(SP3 ASIS)YES, I WOULD LIKE TO USE YOUR REVOLVING CREDIT PLAN.
34 LTD @(SP3 ASIS)YOUR SIGNATURE _____ DATE _____
35 ENDLTD
36 ;
37 IF NUMPY LE 0 REJECT ENDIF
38 LETTER 1
39 ACCEPT

```

Figure 720 Example 4 Print Length for Edited Data Names

GEORGE & CO.
16255 VENTURA BLVD.
ENCINO, CALIFORNIA

11/18/01
60002587

SANTA FE HOSP ASSN
1212 WISCONSIN DRIVE
LOS ANGELES CA 90023

DEAR CUSTOMER:

YOU NOW HAVE THE OPTION OF PAYING YOUR BALANCE IN EASY MONTHLY
INSTALLMENTS BY USING OUR REVOLVING CREDIT PLAN. IF YOU WISH TO
DO SO PLEASE SIGN BELOW AND RETURN IN THE ENCLOSED ENVELOPE.

BALANCE	2,230.17
NO. OF MONTHLY INSTALLMENTS	24
MONTHLY PAYMENT	103.40
PORTION APPLIED TO BALANCE	92.92
INTEREST	9.48

SINCERELY,

GEORGE & CO.

YES, I WOULD LIKE TO USE YOUR REVOLVING CREDIT PLAN.

YOUR SIGNATURE _____ DATE _____

Figure 721 [Example 4](#) Letter Writing Generated Letter

Example 5 Printing a Range of Letter Lines

The LTD command can be followed by a number to associate a letter text statement with a specific line number (for more information, see [LTD Statements and Keywords on page 733](#)). A line number can range from 1 to 99 in value. The same line number can be placed on multiple letter detail lines (LTDs) that are not necessarily consecutive. When a line number is placed on a letter detail line, that line number is propagated to all letter detail lines below it until the next letter detail line that contains a line number.

The range of letter lines to be printed is determined by placing a range of lines on the invoking LETTER statement. The format of that statement is:

```
LETTER nn [{FROM ff TO tt | FROM ff}]  
  where:  
  nn   is the application number  
  ff   is the From range line number  
  tt   is the To range line number
```

An example of this format is shown below:

```
LETTER 1 FROM 2 TO 5
```

The statement in this example invokes LETTER 1 and prints all detail lines that have a line number of 2, 3, 4, or 5 following the LTD keyword.

If all text lines to be invoked have the same line number associated with the LTD command, the statement would be:

```
LETTER 1 FROM 2 TO 2
```

This LETTER command format invokes the text (one or more lines) associated with line number 2 only.

When printing a range of pages, page ejection occurs automatically when the first physical line of the letter is printed. The automatic page ejection can be eliminated by specifying NOEJECT in the header (LTH) statement.

VISION:Results generates and can dynamically alter a special reserved word field (DYLCNxxLNC), which is the letter writing function's line counter for letter writing application xx. When DYLCNxxLNC becomes larger than the maximum lines per page, VISION:Results forces page ejection. Consequently, if the program dynamically changes DYLCNxxLNC to a value that is greater than the maximum lines per page, page ejection occurs when LETTER xx is next invoked.

The From-To Line Range technique makes the printing of repetitive information formats easy to handle. The program shown in [Figure 722](#) demonstrates how repetitive billing information can be printed using the letter writing function.

Line number 1 encompasses letter statements 24–35. Line number 2 is letter statement 36, and line number 3 encompasses letter statements 37 and 38. For each letter, line number 1 is started once using program statement 41; line number 2 is invoked multiple times using statement 49; and line number 3 is invoked once using statement 55. One of the letters generated by this program is shown in [Figure 723](#).

```
1  FILE SMGFILE VB 1000 6000
2  OSMAS41      1000 1          ; A/R OBLIGATION SUMMARY
3  OSNAME       29 1          ; NAME
4  OSADR1       22 30          ; ADDRESS 1.
5  OSADR2       22 96          ; CITY, STATE AND ZIP
6  OSBALANCE    6 118 PD 2
7  OSMAS41      818 124
8  OSACCT NO    13 124          ; ACCOUNT NUMBER
9  OSNO TRLS    2 200 PD E      ; NUMBER OF TRAILERS
10 OSTRANAREA   740 202         ; OCCURS 20 TIMES
11 REDEFINE AT OSTRANAREA       ; TRAILER LENGTH = 37 BYTES
12 OSOBLGAMT    5 213 PD 2 E    ; OBLIGATION AMOUNT
13 OSDUE_DATE   4 228 PD D      ; DUE DATE
14 ;
15 WORKAREA
16 CNT          2          PD
17 WK_AMT       5          PD 2
18 WK_DUE_DT    7 21
19 WK_DUE_DT2   6 22
20 WK_YY_MM     4 22
21 WK_DD        2 26
22 ;
23 LTH 01 66 WIDE
24 LTD1 @23 GEORGE & CO.
25 LTD @23 17418 CHATSWORTH ST.
26 LTD @23 GRANADA HILLS, CALIFORNIA 91344
```

Figure 722 [Example 5](#) Letter Writing Input Statements (Page 1 of 2)

```

27 LTD  @ (SP2) @50 @DYLDATE
28 LTD  @ (SP2)@OSNAME @47 @OSACCT_NO(B)
29 LTD  @ (ASIS)@OSADR1
30 LTD  @ (ASIS)@OSADR2
31 LTD  @ (SP2 ASIS)DEAR CUSTOMER:
32 LTD  @ (SP2) THE BALANCE OF YOUR ACCOUNT IS DUE. PLEASE REMIT
33 LTD  $@OSBALANCE(A) BEFORE DECEMBER 20, 1999.
34 LTD  @ (SP2) @8 DUE DATE @55 AMOUNT
35 LTD  @ (SP2)
36 LTD2 @9 @WK DUE DT2(D) @49 @WK_AMT(A)
37 LTD3 @ (SP2) @50 SINCERELY,
38 LTD  @ (SP4) @50 GEORGE & CO.
39 ENDLTD 40 MAIN:
41 LETTER 1 FROM 1 TO 1
42 MOVE OSNO_TRLS TO CNT
43 MOVE 0 TO INW
44 ;
45 LOOP1:
46 IF CNT LE 0 GOTO LOOP9 ENDIF
47 MOVE OSDUE_DATE(INW) TO WK_DUE_DT
48 MOVE OSOBLGAMT(INW) TO WK_AMT
49 LETTER 1 FROM 2 TO 2
50 INW = INW + 37
51 CNT = CNT - 1
52 GOTO LOOP1
53 ;
54 LOOP9:
55 LETTER 1 FROM 3 TO 3
56 ACCEPT

```

Figure 722 [Example 5](#) Letter Writing Input Statements (Page 2 of 2)

```

GEORGE & CO.
17418 CHATSWORTH ST.
GRANADA HILL, CALIFORNIA 91344

11/18/01

MARY ANNE BLACK
BOX 404 MOUNTAIN RD
GREENWICH NY 12834
1013454541

DEAR CUSTOMER:

THE BALANCE OF YOUR ACCOUNT IS DUE. PLEASE REMIT $89.09 BEFORE
DECEMBER 20, 1998.

DUE DATE          AMOUNT

05/04/01          21.09
04/28/01          18.00
03/31/01          27.96
02/27/01          22.04

SINCERELY,

GEORGE & CO.

```

Figure 723 [Example 5](#) Letter Writing Generated Letter

Example 6 Printing Using the Laser Option

George & Co.'s marketing department has prepared a catalog of special items that they want to make available to their best customers. These customers all have an account code of MO on the accounts receivable file. The marketing manager wants a cover letter sent to all MO accounts, offering the catalog and a copy of the guide Restaurants of Southern California to all who respond by September 16th.

This example differs from [Selecting Records and Generating Reports on page 750](#) in that character sets are changed within the letter and the output is laser printed.

```
1  COPY DYAUDREC
2  LTH 01 66 WIDE LASER
3  LTD @23 GEORGE & CO.
4  LTD @23 16255 VENTURA BLVD.
5  LTD @23 ENCINO, CALIFORNIA
6  LTD @(SP2) @50 @DYLDATE
7  LTD @(SP2)@NAME @50 @ACCTNO
8  LTD @(ASIS)@ADD1
9  LTD @(ASIS)@ADD2
10 LTD @(SP2 ASIS)DEAR CUSTOMER:
11 LTD @(SP2 ASIS) @25 @(CS2) FREE OFFER
12 LTD LTD @(SP2 CS1)AS A VALUED CUSTOMER, YOU CAN RECEIVE OUR SPECIAL
13 LTD PURCHASE CATALOG IF YOU RESPOND BY SEPTEMBER 16. JUST
14 LTD RETURN THE FORM BELOW AND THE CATALOG WILL BE ON ITS WAY.
15 LTD TOGETHER WITH A FREE COPY OF
16 LTD @(CS2) '"RESTAURANTS OF SOUTHERN CALIFORNIA."'
17 LTD @(SP2 CS1) @50 SINCERELY,
18 LTD @(SP4) @50 GEORGE & CO.
19 LTD @(SP2) @23 -----
20 LTD @(SP2 ASIS)PLEASE SEND MY CATALOG AND BONUS GIFT TO:
21 LTD @(SP2 ASIS)NAME: _____
22 LTD @(SP2 ASIS)ADDRESS: _____
23 LTD @(SP2 ASIS)CITY: _____ STATE: _____ ZIP: _____
24 ENDLTH
25 IF ACCTCODE EQ 'MO'
26   LETTER 01
27   ELSE REJECT
28 ENDIF
```

Figure 724 Example 6 Letter Writing Input Statements

As part of the LTH header in statement 2, the laser option has been specified that sets an indicator to determine the maximum number (nn) of character sets per letter. Because there is no (nn) coded, the default is four separate/unique character sets within one letter.

The (CSnn) keywords on LTD statements 11, 12, 16, and 17 are used to specify which character set is to be printed. The LTD statements 11 and 16 use the (CS2) keyword, which specifies that the second character set specified in the JCL is to be used. The (CS1) keyword on the LTD statements 12 and 17 resets the character set to its original position.

```

                                GEORGE & CO.
                                16255 VENTURA BLVD.
                                ENCINO, CALIFORNIA

                                07/07/01

MORENO, ELIZABETH                6039804
208 1/2 INEZ ST
NORWALK CA 90605

DEAR CUSTOMER:

                                FREE OFFER

AS A VALUED CUSTOMER, YOU CAN RECEIVE OUR SPECIAL PURCHASE CATALOG
IF YOU RESPOND BY SEPTEMBER 16. JUST RETURN THE FORM BELOW AND
THE CATALOG WILL BE ON ITS WAY, TOGETHER WITH A FREE COPY OF
"RESTAURANTS OF SOUTHERN CALIFORNIA."

                                SINCERELY,

                                GEORGE & CO.
                                -----

PLEASE SEND MY CATALOG AND BONUS GIFT TO:
NAME: _____
ADDRESS: _____
CITY: _____ STATE: _____ ZIP: _____

```

Figure 725 [Example 6](#) Letter Writing Generated Letter

Example 7 Generating Two-Up Letters Using the Defaults

This example shows how to produce two-up letters using some of the default values. The phrase two-up means that two letters are produced side-by-side on a page (one on the left, one on the right).

To produce two-up letters, the keyword TWOUP is used in the LTH statement. The keyword BETWEEN (for spaces between letters) is not used, so the default is 5. This means that there are five blank spaces between the left-side letter and the right-side letter on the page. The keyword LEFTSAVE (for the amount of space to hold the left-side letter) is not used, so the default is 10K.

The width of each letter is specified as 60 using the keyword WIDE. The number of lines per page has been specified as 56 using the keyword LONG. The NEWPARA I5 parameter in the LTD statement after DEAR CUSTOMER: causes the entire paragraph to be indented five spaces. The NEWPARA L3 parameter in the LTD statement for the second paragraph causes the first line of the paragraph to be indented three spaces.

The generated letters are shown following the source statements.

```

COPY DYAUDREC
LTH 1 60 WIDE 56 LONG TWOUP
LTD @(SP2) @25
LTD @25
LTD @25
LTD @(SP2) @50
LTD @(SP2) @1 @NAME @50 @DYLDATE
LTD @1 @ADD1 @ACCTNO(Z)
LTD @1 @ADD2
LTD @(SP2) @1 DEAR CUSTOMER:
LTD @(SP2 NEWPARA I5) THIS
LTD FORM IS BEING SENT TO YOU SO THAT OUR INDEPENDENT AUDITORS
LTD CAN CONFIRM THE CORRECTNESS OF OUR RECORDS. THIS IS NOT A
LTD REQUEST FOR PAYMENT. OUR RECORDS ON @TRANDATE(D), SHOWED
LTD A BALANCE OF $@BALANCE(A).
LTD @(SP2 NEWPARA L3) PLEASE
LTD CONFIRM WHETHER THIS AGREES WITH YOUR RECORDS ON THAT DATE
LTD BY SIGNING AND RETURNING THIS FORM DIRECT TO OUR AUDITORS, RYAN,
LTD RYAN AND COMPANY, 201 SOUTH HOPE STREET, LOS ANGELES,
LTD CALIFORNIA 90071. AN ADDRESSED ENVELOPE IS ENCLOSED FOR THIS
LTD PURPOSE. IF YOU FIND ANY DIFFERENCE, PLEASE REPORT THEM TO
LTD THE AUDITORS.
LTD @(SP2) @40 VERY TRULY YOURS,
LTD @(SP3) @40 CONTROLLER
LTD @40 GEORGE & CO.
LTD @(SP5) @25 C O N F I R M A T I O N
LTD @25 - - - - -
LTD @(SP2) @1 RYAN, RYAN AND COMPANY
LTD @1 201 SOUTH HOPE STREET
LTD @1 LOS ANGELES, CA 90071
LTD @(SP2) @1 THE ABOVE INFORMATION IS CORRECT EXCEPT AS NOTED BELOW:
LTD @(SP2) @1 -----
LTD @(SP2) @1 -----
LTD @(SP2) @1 BY: ----- DATE: -----
ENDLTD

LETTER 1

```

Figure 726 [Example 7](#) Letter Writing Input Statements

GEORGE & CO. 16255 VENTURA BLVD. ENCINO, CA	GEORGE & CO. 16255 VENTURA BLVD. ENCINO, CA
3/24/01	3/24/01
8006547	6208657
TORRES, ERNESTO 23444 PARK LANE LOS ANGELES CA	CHO PYUNG, SUM 33333 PALL MALL GLENDALE CA
DEAR CUSTOMER:	DEAR CUSTOMER:
THIS FORM IS BEING SENT TO YOU SO THAT OUR INDEPENDENT AUDITORS CAN CONFIRM THE CORRECTNESS OF OUR RECORDS. THIS IS NOT A REQUEST FOR PAYMENT. OUR RECORDS ON 3/15/01, SHOWED A BALANCE OF \$44.99.	THIS FORM IS BEING SENT TO YOU SO THAT OUR INDEPENDENT AUDITORS CAN CONFIRM THE CORRECTNESS OF OUR RECORDS. THIS IS NOT A REQUEST FOR PAYMENT. OUR RECORDS ON 3/17/01, SHOWED A BALANCE OF \$32.00.
PLEASE CONFIRM WHETHER THIS AGREES WITH YOUR RECORDS ON THAT DATE BY SIGNING AND RETURNING THIS FORM TO OUR AUDITORS, RYAN, RYAN AND COMPANY, 201 SOUTH HOPE STREET, LOS ANGELES, CALIFORNIA 90071. AN ADDRESSED ENVELOPE IS ENCLOSED FOR THIS PURPOSE. IF YOU FIND ANY DIFFERENCE, PLASE REPORT THEM TO THE AUDITORS.	PLEASE CONFIRM WHETHER THIS AGREES WITH YOUR RECORDS ON THAT DATE BY SIGNING AND RETURNING THIS FORM TO OUR AUDITORS, RYAN, RYAN AND COMPANY, 201 SOUTH HOPE STREET, LOS ANGELES, CALIFORNIA 90071. AN ADDRESSED ENVELOPE IS ENCLOSED FOR THIS PURPOSE. IF YOU FIND ANY DIFFERENCE, PLASE REPORT THEM TO THE AUDITORS.
VERY TRULY YOURS,	VERY TRULY YOURS,
CONTROLLER GEORGE & CO.	CONTROLLER GEORGE & CO.
CONFIRMATION -----	CONFIRMATION -----
RYAN, RYAN AND COMPANY 201 SOUTH HOPE LOS ANGELES, CA 90071	RYAN, RYAN AND COMPANY 201 SOUTH HOPE LOS ANGELES, CA 90071
THE ABOVE INFORMATION IS CORRECT EXCEPT AS NOTED BELOW:	THE ABOVE INFORMATION IS CORRECT EXCEPT AS NOTED BELOW:
_____	_____
_____	_____
BY: _____ DATE: _____	BY: _____ DATE: _____

Figure 727 [Example 7](#) Letter Writing Generated Letter

Example 8 Generating Two-Up Letters Using DYLCNxxSW

This example uses a two-page letter. By using the switching field DYLCNxxSW, the first page of the letter for a record and the first page of the letter for the next record are printed side-by-side on a single page; the second page of the letter for the first record and the second page for the next record are printed side-by-side on the following page, as shown in [Figure 730](#). Without using the switch, the two pages of the letter would be printed side-by-side for each record, as shown in [Figure 728](#).

In the program shown in [Figure 729](#), the keyword TWOUP specifies two-up letters. The keyword SWITCH specifies page switching. The @EJECT parameter in the first LTD statement of the second letter specifies a page eject. The logic statements specify using only the first 10 records of the file (for this example). The other logic statements specify page switching by moving a Y into DYLCNxxSW before the next letter is executed. In other words, left-to-right page

switching occurs only after the next record (for a pair) is read. The letter records for this next record are printed on the right side of the page. This example uses the default value of 10K to hold the letter records of the left side of the page. You can specify your own value for this parameter by using the keyword LEFTSAVE.

GEORGE & CO. 16255 VENTURA BLVD. ENCINO, CA	PAGE 2	GEORGE & CO.
	3/24/01	3/24/01
	8006547	8006547
TORRES, ERNESTO 23444 PARK LANE LOS ANGELES CA	TORRES, ERNESTO 23444 PARK LANE LOS ANGELES CA	
DEAR CUSTOMER:	IF YOUR RECORDS DO NOT AGREE WITH OURS, PLEASE NOTE THE DISCREPANCIES, SIGN AND RETURN THE FORM BELOW.	
THIS FORM IS BEING SENT TO YOU SO THAT OUR INDEPENDENT AUDITORS CAN CONFIRM THE CORRECTNESS OF OUR RECORDS. THIS IS NOT A REQUEST FOR PAYMENT. OUR RECORDS ON 3/15/01, SHOWED A BALANCE OF \$44.99.	VERY TRULY YOURS,	
PLEASE CONFIRM WHETHER THIS AGREES WITH YOUR RECORDS ON THAT DATE BY SIGNING AND RETURNING THIS FORM TO OUR AUDITORS, RYAN, RYAN AND COMPANY, 201 SOUTH HOPE STREET, LOS ANGELES, CALIFORNIA 90071. AN ADDRESSED ENVELOPE IS ENCLOSED FOR THIS PURPOSE. IF YOU FIND ANY DIFFERENCE, PLEASE REPORT THEM TO THE AUDITORS.	CONTROLLER GEORGE & CO.	
	DISCREPANCIES -----	
	RYAN, RYAN AND COMPANY 201 SOUTH HOPE LOS ANGELES, CA 90071	
CONFIRMATION -----	THE ABOVE INFORMATION IS INACCURATE AS NOTED BELOW:	
RYAN, RYAN AND COMPANY 201 SOUTH HOPE LOS ANGELES, CA 90071	_____	
THE ABOVE INFORMATION IS CORRECT EXCEPT AS NOTED BELOW:	_____	
_____	BY: _____ DATE: _____	
BY: _____ DATE: _____		

Figure 728 [Example 8](#) Letter Writing Generated Letter Printing

GEORGE & CO. 16255 VENTURA BLVD. ENCINO, CA 3/24/01 8006547 TORRES, ERNESTO 23444 PARK LANE LOS ANGELES CA DEAR CUSTOMER: THIS FORM IS BEING SENT TO YOU SO THAT OUR INDEPENDENT AUDITORS CAN CONFIRM THE CORRECTNESS OF OUR RECORDS. THIS IS NOT A REQUEST FOR PAYMENT. OUR RECORDS ON 3/15/01, SHOWED A BALANCE OF \$44.99. PLEASE CONFIRM WHETHER THIS AGREES WITH YOUR RECORDS ON THAT DATE BY SIGNING AND RETURNING THIS FORM TO OUR AUDITORS, RYAN, RYAN AND COMPANY, 201 SOUTH HOPE STREET, LOS ANGELES, CALIFORNIA 90071. AN ADDRESSED ENVELOPE IS ENCLOSED FOR THIS PURPOSE. IF YOU FIND ANY DIFFERENCE, PLEASE REPORT THEM TO THE AUDITORS. VERY TRULY YOURS, CONTROLLER GEORGE & CO.	GEORGE & CO. 16255 VENTURA BLVD. ENCINO, CA 3/24/01 6208657 CHO PYUNG, SUM 33333 PALL MALL GLENDALE CA DEAR CUSTOMER: THIS FORM IS BEING SENT TO YOU SO THAT OUR INDEPENDENT AUDITORS CAN CONFIRM THE CORRECTNESS OF OUR RECORDS. THIS IS NOT A REQUEST FOR PAYMENT. OUR RECORDS ON 3/17/01, SHOWED A BALANCE OF \$32.00. PLEASE CONFIRM WHETHER THIS AGREES WITH YOUR RECORDS ON THAT DATE BY SIGNING AND RETURNING THIS FORM TO OUR AUDITORS, RYAN, RYAN AND COMPANY, 201 SOUTH HOPE STREET, LOS ANGELES, CALIFORNIA 90071. AN ADDRESSED ENVELOPE IS ENCLOSED FOR THIS PURPOSE. IF YOU FIND ANY DIFFERENCE, PLEASE REPORT THEM TO THE AUDITORS. VERY TRULY YOURS, CONTROLLER GEORGE & CO.
C O N F I R M A T I O N ----- RYAN, RYAN AND COMPANY 201 SOUTH HOPE LOS ANGELES, CA 90071 THE ABOVE INFORMATION IS CORRECT EXCEPT AS NOTED BELOW: _____ _____ BY: _____ DATE: _____ PAGE 2 GEORGE & CO.	C O N F I R M A T I O N ----- RYAN, RYAN AND COMPANY 201 SOUTH HOPE LOS ANGELES, CA 90071 THE ABOVE INFORMATION IS CORRECT EXCEPT AS NOTED BELOW: _____ _____ BY: _____ DATE: _____ PAGE 2 GEORGE & CO.
3/24/01 8006547 TORRES, ERNESTO 23444 PARK LANE LOS ANGELES CA IF YOUR RECORDS DO NOT AGREE WITH OURS, PLEASE NOTE THE DISCREPANCIES, SIGN AND RETURN THE FORM BELOW. VERY TRULY YOURS, CONTROLLER GEORGE & CO.	3/24/01 6208657 CHO PYUNG, SUM 33333 PALL MALL GLENDALE CA IF YOUR RECORDS DO NOT AGREE WITH OURS, PLEASE NOTE THE DISCREPANCIES, SIGN AND RETURN THE FORM BELOW. VERY TRULY YOURS, CONTROLLER GEORGE & CO.

Figure 730 Example 8 Letter Writing Generated Letter Printing Side-by-Side with Switch (Page 1 of 2)

DISCREPANCIES ----- RYAN, RYAN AND COMPANY 201 SOUTH HOPE LOS ANGELES, CA 90071 THE ABOVE INFORMATION IS INACCURATE AS NOTED BELOW: _____ _____ BY: _____ DATE: _____	DISCREPANCIES ----- RYAN, RYAN AND COMPANY 201 SOUTH HOPE LOS ANGELES, CA 90071 THE ABOVE INFORMATION IS INACCURATE AS NOTED BELOW: _____ _____ BY: _____ DATE: _____

Figure 730 Example 8 Letter Writing Generated Letter Printing Side-by-Side with Switch (Page 2 of 2)

Example 9 Writing Letters to an Output File

Letter records (lines of a letter) can be written to an output file of your own choosing. If no output file is specified, the default file is AUDPRINT/SYS009. There are several reasons for using your own output file. Some of these are:

- If your letters are longer than 131 characters, you cannot use AUDPRINT/SYS009.
- If your letters require using ASA carriage control characters (rather than machine code), you cannot use AUDPRINT/SYS009.
- To manipulate the letter records independently of other records for audit functions such as linear regression, make the letter records independent from AUDPRINT/SYS009.

In this example, the first two items listed above are the reason for writing letter records to an output file. The letter is 200 characters wide and requires ASA carriage control. The number of lines per page is 56. The output letter records are written to the file with the DD/DLBL name of DYLTRAB. The default blocking factor for the output file is 10. The keyword ASA specifies ASA carriage control characters for the output letter records.

```

COPY DYAUDREC
LTH 1 200 WIDE 56 LONG DYLTRAB ASA
LTD @(SP2) @25
LTD @25
LTD @25
LTD @(SP2) @50
                                GEORGE & CO.
                                16255 VENTURA BLVD.
                                ENCINO, CALIFORNIA
                                @DYLDATE

LTD @(SP2) @1    @NAME                                @50    @ACCTNO(Z)

LTD @1            @ADD1
LTD @1            @ADD2
LTD @(SP2) @1    DEAR CUSTOMER:
LTD @(SP2 NEWPARA L3) THIS
LTD FORM IS BEING SENT TO YOU SO THAT OUR INDEPENDENT AUDITORS
LTD CAN CONFIRM THE CORRECTNESS OF OUR RECORDS. THIS IS NOT A
LTD REQUEST FOR PAYMENT. OUR RECORDS ON @TRANDATE(D), SHOWED
LTD A BALANCE OF $@BALANCE(A).
LTD @(SP2 NEWPARA L3) PLEASE
LTD CONFIRM WHETHER THIS AGREES WITH YOUR RECORDS ON THAT DATE
LTD BY SIGNING AND RETURNING THIS FORM DIRECT TO OUR AUDITORS, RYAN,
LTD RYAN AND COMPANY, 201 SOUTH HOPE STREET, LOS ANGELES,
LTD CALIFORNIA 90071. AN ADDRESSED ENVELOPE IS ENCLOSED FOR THIS
LTD PURPOSE. IF YOU FIND ANY DIFFERENCE, PLEASE REPORT THEM TO
LTD THE AUDITORS.
LTD @(SP2) @40                                VERY TRULY YOURS,
LTD @40                                CONTROLLER
LTD @40                                GEORGE & CO.
LTD @(SP5) @25    C O N F I R M A T I O N
LTD @25    - - - - -
LTD @(SP2) @1    RYAN, RYAN AND COMPANY
LTD @1 201 SOUTH HOPE STREET
LTD @1 LOS ANGELES, CA 90071
LTD @(SP2) @1    THE ABOVE INFORMATION IS CORRECT EXCEPT AS NOTED BELOW:
LTD @(SP2) @1    -----
LTD @(SP2) @1    -----
LTD @(SP2) @1 BY: ----- DATE: -----
ENDLTD

LETTER 1

```

Figure 731 [Example 9](#) Letter Writing Input Statements

Chapter 29: Using Diagrams and Analysis

This chapter describes scatter diagrams, linear regression analysis, and trend line analysis, including an example and implementation.

Defining Scatter Diagrams, Linear Regression Analysis, and Trend Line Analysis

Scatter diagrams, linear regression analysis, and trend line analysis are methods for studying and measuring the relationship between two variables when data concerning the two variables are compiled empirically.

- Use scatter diagrams to graphically show the relationship between the two variables.
- Use linear regression analysis to define a theoretical straight line that ties the dependent variable to the independent variable.
- Trend line analysis are a subset of linear regression analysis in which the independent variable is time-dependent.

Scatter Diagram

A scatter diagram is the plotting of a dependent variable against an independent variable in a graphic representation. Each set of X,Y values defines a point on the graph. For example, [Figure 732 on page 768](#) shows the following variable information plotted in a scatter diagram.

Variable	Values								
X	1	4	5	7	9	10	12	15	18
Y	5	11	13	15	21	25	30	35	40

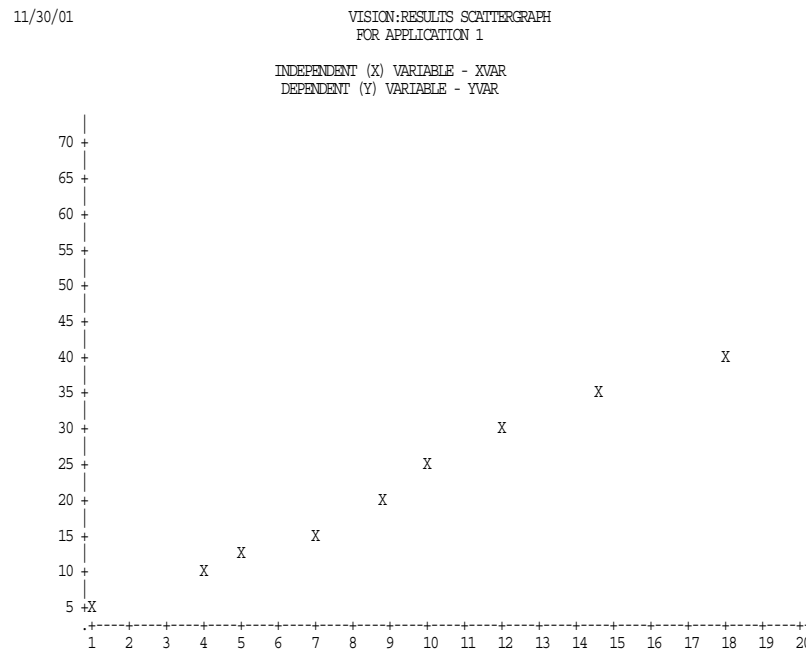


Figure 732 Scatter Diagram

Linear Regression Analysis

Use linear regression analysis to define the equation of a straight line by which the dependent variable can be estimated from the independent variable. A straight line is defined by the following equation:

$$Y = a + bX$$

where:

Y is the dependent variable.

a is the Y intercept (the point at which the line intersects the Y-axis).

b is the slope of the line.

X is the independent variable.

Using the method of least squares, a and b above are calculated so that the straight line that is defined represents the line that fits best through the actual points. For example, for the information shown above and illustrated in [Figure 732](#), the linear regression analysis line is:

$$Y = 2.2938 + 2.1525X$$

[Figure 733 on page 769](#) shows the straight line in the scatter diagram. The dependent value Y can now be estimated by the independent value X using the above equation. Therefore, if X=2, then Y=6.5988.

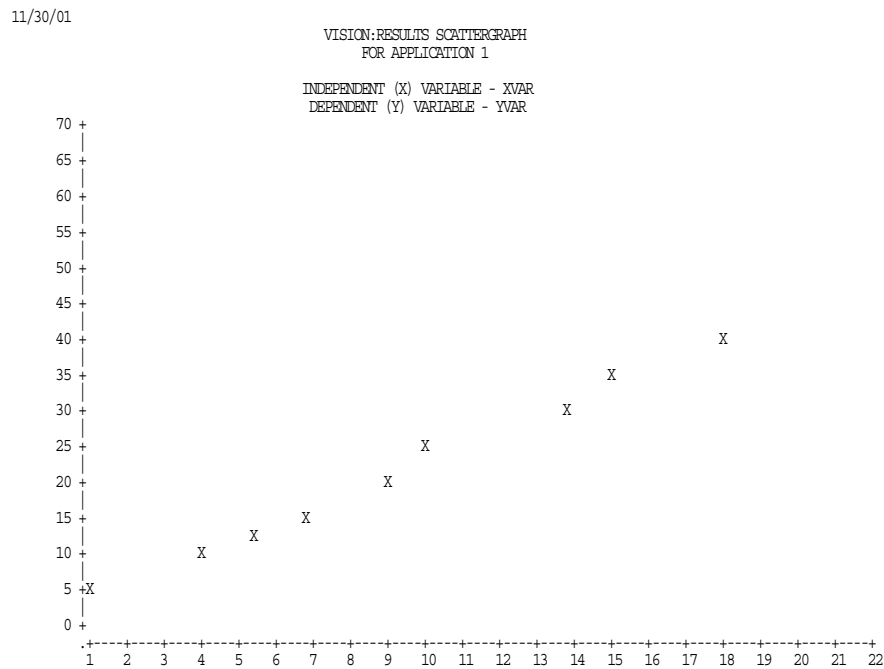


Figure 733 Linear Regression Analysis Line

Because the straight line approximates the positioning of the actual data, the prediction of the dependent variable as a function of the independent variable is subject to a degree of inaccuracy. The amount of that inaccuracy can be estimated within confidence limits determined by the numerical value of the standard error of the estimate.

The standard error of the estimate has the same relationship to the regression line as the standard deviation does to the arithmetic mean. The normal curve Z values are used in much the same way. That is, 68.26% of the deviations are within one standard error of the estimate from the regression line; 95.44% of the deviations are within two standard errors of the regression line.

For example, the standard error of the estimate in the above data is 1.1789. The predicted dependent value for the independent value 4 is 10.9038. With a 95.44% confidence, you can be certain that the actual dependent value is in the range 10.9038 ± 2.3578 , or between 8.5460 and 13.2626.

Correlation Coefficient

The Correlation Coefficient is a numerical measure of the amount of relationship between the two variables. The value of the correlation coefficient ranges from -1 to +1. A value of 0 indicates that there is no linear relationship between the two variables. When the value is positive, the Y values increase as the X values increase (the regression line slopes upward from left to right). When the value is negative, the Y values decrease as the X values increase (the regression line slopes downward from left to right).

Coefficient of Determination

The Coefficient of Determination indicates how well the regression line approximates the distribution of points. Its value may range from 0 to 1. The closer it is to 0, the less likely the two variables are linearly related. The closer it is to 1, the closer the distribution of points is to the regression line.

Coefficient of Non-Determination

The Coefficient of Non-Determination is 1 minus the coefficient of determination. The closer it is to 1, the less likely the two variables are linearly related. The closer it is to 0, the closer the distribution of points is to the regression line.

VISION:Results automatically calculates the slope and Y intercept of the regression line, the standard error of the estimate, the correlation coefficient, the coefficient of determination, and the coefficient of non-determination.

Trend Line Analysis

Trend line analysis as implemented in VISION:Results are a subfunction of linear regression analysis in which the dependent variable is time-oriented (for example, months or years).

Further Sources of Information

This chapter is a brief overview of statistical analysis. For more information, refer to an elementary statistics textbook.

The sections that follow describe the formats to use when requesting scatter diagrams, linear regression analysis, and/or trend line analysis plotting the dependent variable against the independent variable for a graphical representation of the relationship between the two.

Input into VISION:Results

The VISION:Results FILE and field definition statements for scatter diagrams, linear regression analysis, and trend line analysis are identical except for the first keyword. In all cases, the field definition statements must precede the statement defining your scatter diagram, linear regression analysis, or trend line analysis request. The format of these statements is shown in [Figure 734](#) through [Figure 736](#).

```
SCATTER nn XNAME dataname YNAME dataname
[TITLE dataname]
XSCATTER [n] FROM nnnnnnnnnnnnnn BY nnnnnnnnnnnnnn [FOR nn]
YSCATTER [n] FROM nnnnnnnnnnnnnn BY nnnnnnnnnnnnnn [FOR nn]
```

Figure 734 Syntax for SCATTER Statement

```
LINEAR nn XNAME dataname YNAME dataname
[TITLE dataname]
XSCATTER [n] FROM nnnnnnnnnnnnnn BY nnnnnnnnnnnnnn [FOR nn]
YSCATTER [n] FROM nnnnnnnnnnnnnn BY nnnnnnnnnnnnnn [FOR nn]
```

Figure 735 Syntax for LINEAR Statement

```
TREND nn XNAME dataname YNAME dataname
[TITLE dataname]
XSCATTER [n] FROM nnnnnnnnnnnnnn BY nnnnnnnnnnnnnn [FOR nn]
YSCATTER [n] FROM nnnnnnnnnnnnnn BY nnnnnnnnnnnnnn [FOR nn]
```

Figure 736 Syntax for TREND Statement

SCATTER in Figure 734 only	Required. Indicates that a scatter diagram is requested. No abbreviations are allowed. The keywords LINEAR or TREND can be used instead of SCATTER. The three terms are interchangeable.
LINEAR in Figure 735 only	Required. Describes the application as a linear regression analysis. No abbreviations are allowed. The keywords TREND or SCATTER can be used instead of LINEAR. The three terms are interchangeable.
TREND in Figure 736 only	Required. Describes the application as trend line analysis. No abbreviations are allowed. The keywords LINEAR and SCATTER can be used instead of TREND. The three terms are interchangeable.
nn	Required. A unique application number must be assigned to each request, even if different commands are used in the defining statements. This number must be an integer with a value from 1 to 99.
XNAME	Required. The data name of the field to be used as the independent (X) variable. Must be previously defined in a field definition statement. Any valid data name can be used.

YNAME	Required. The data name of the field to be used as the dependent (Y) variable. Must be previously defined in a field definition statement. Any valid data name can be used.
TITLE	The data name of the field containing the report title presented as a literal. The contents of the title field print at the top of the linear regression analysis (and scatter diagram, if also requested). Must be previously defined in a field definition statement. Any valid data name can be used. The maximum size of the field is 70 characters.

Note: The remaining fields are required only if you are requesting a scatter diagram.

XSCATTER	Required. This value is the scaling factor for the independent variable (X-axis). The values on the X-axis are divided by this scaling factor before being printed. If omitted, this value defaults to 1 unit. <table><tr><td>blank</td><td>1 unit</td></tr><tr><td>1</td><td>Thousands</td></tr><tr><td>2</td><td>Millions</td></tr><tr><td>3</td><td>Billions</td></tr><tr><td>4</td><td>Trillions</td></tr></table>	blank	1 unit	1	Thousands	2	Millions	3	Billions	4	Trillions
blank	1 unit										
1	Thousands										
2	Millions										
3	Billions										
4	Trillions										
FROM	Required. The FROM starting value is on the X-axis of the scatter diagram. It is a numeric value. Negative values must be preceded by a minus sign. A leading dollar sign, a floating decimal point, and commas are allowed. If a dollar sign and a minus sign are both used, the dollar sign must precede the minus sign; therefore, \$- is correct. If no decimal point is specified, right-alignment is assumed. The value (including digits, any dollar sign, any minus sign, any decimal point, and any commas) cannot exceed 14 characters. The unit value of this field is determined by the XSCATTER scaling factor. Therefore, if 1 is coded following XSCATTER, 1.5 coded after FROM represents 1,500.										
BY	Required. The BY value is the unit value on the X-axis of the scatter diagram. This must be a positive numeric value. A leading dollar sign, a floating decimal point, and commas are allowed. The value (including digits, any dollar sign, any decimal point, and any commas) cannot exceed 14 characters. The unit value of this field is determined by the XSCATTER scaling factor. Therefore, if 2 is coded following XSCATTER, 4.15 coded after BY represents 4.15 million.										
FOR	This value is the number of intervals to be printed on the X-axis of the scatter diagram. The value of this field must be an integer between 1 and 30, inclusive. If omitted, this value defaults to 30.										

YSCATTER	Required. This value is the scaling factor for the dependent variable (Y-axis). The values on the Y-axis are divided by this scaling factor before being printed. If omitted, this value defaults to 1 unit. <table> <tr> <td>blank</td><td>1 unit</td></tr> <tr> <td>1</td><td>Thousands</td></tr> <tr> <td>2</td><td>Millions</td></tr> <tr> <td>3</td><td>Billions</td></tr> <tr> <td>4</td><td>Trillions</td></tr> </table>	blank	1 unit	1	Thousands	2	Millions	3	Billions	4	Trillions
blank	1 unit										
1	Thousands										
2	Millions										
3	Billions										
4	Trillions										
FROM	Required. The FROM starting value is on the Y-axis of the scatter diagram. It is a numeric value. Negative values must be preceded by a minus sign. A leading dollar sign, a floating decimal point, and commas are allowed. If a dollar sign and a minus sign are both used, the dollar sign must precede the minus sign; therefore, \$- is correct. If no decimal point is specified, right-alignment is assumed. The value (including digits, any dollar sign, any minus sign, any decimal point, and any commas) cannot exceed 14 characters. The unit value of this field is determined by the YSCATTER scaling factor. Therefore, if 1 is coded following YSCATTER, 1.5 coded after FROM represents 1,500.										
BY	Required. The BY value is the unit value on the Y-axis of the scatter diagram. This must be a positive numeric value. A leading dollar sign, a floating decimal point, and commas are allowed. The value (including digits, any dollar sign, any decimal point, and any commas) cannot exceed 14 characters. The unit value of this field is determined by the YSCATTER scaling factor. Therefore, if 2 is coded following YSCATTER, 4.15 coded after BY represents 4.15 million.										
FOR	This value is the number of intervals to be printed on the Y-axis of the scatter diagram. The value of this field must be an integer between 1 and 20, inclusive. If omitted, this value defaults to 20.										

Regression Command

To use linear regression analysis, you must specify the REGRESSION command in your program procedure logic. If this command is omitted, no regression analysis is performed.

REGRESSION nn

The REGRESSION command and the application number nn are directly associated with the LINEAR, TREND, and/or SCATTER definition statements. The application numbers for each request within a single VISION:Results program must be unique, even if you use different keywords in your defining statements.

For example, if a single program included requests for a linear regression analysis and a trend line analysis with a scatter diagram, you must specify two unique application numbers, one for each report, as shown in [Figure 737](#) and [Figure 738](#).

```
LINEAR 01 XNAME SALES YNAME PROFIT

TREND 02 XNAME YEARS YNAME SALES TITLE HDRTITLE
        XSCATTER FROM 1998 BY 1 FOR 12
        YSCATTER 1 FROM 0 BY 100 FOR 20
```

Figure 737 Regression Command, Example 1

When you are requesting a scatter diagram, linear regression analysis, or trend line analysis, you can use the keywords LINEAR, TREND, and SCATTER interchangeably in your request statements. The command to use in your procedure logic is REGRESSION; this command is the same for all three applications. You can also specify the type of regression analysis (LINEAR, TREND, or SCATTER) following the REGRESSION command.

```
REGRESSION LINEAR 01
REGRESSION TREND 02
REGRESSION SCATTER 03
```

Figure 738 Regression Command, Example 2

If the REGRESSION command is omitted, no scatter diagram, linear regression analysis, or trend line analysis is performed or produced, even though the application is defined in your program.

[Figure 739](#) shows the VISION:Results statements required to generate a linear regression analysis and scatter diagram to determine if a linear relationship exists between the values in the fields SALES and PROFIT on the file, INFILE.

```
FILE INFILE                                1
  SALES      8      1      PD      2      2
  PROFIT     8      PD      2      3

LINEAR 01 XNAME SALES YNAME PROFIT        4
  XSCATTER 2 FROM 0 BY .5 FOR 24          5
  YSCATTER 2 FROM 0 BY .2 FOR 18          6

REGRESSION LINEAR 01                      7
```

Figure 739 Linear Regression Analysis and Scatter Diagram Input Statements

The numbers on the right are used to explain the VISION:Results statements.

Statement 1 is the FILE statement, defining INFILE as the input file. Statements 2 and 3 are the field definition statements, defining the fields that contain the independent data (SALES) and the dependent data (PROFIT).

The FILE and field definition statements must precede the statement defining your scatter diagram, linear regression analysis, and/or trend line analysis application(s).

Statements 4–6 are requesting linear regression analysis and a scatter diagram. The fields defined by the data names SALES and PROFIT are used as independent and dependent variables, respectively.

For the scatter diagram parameters, a scaling factor of 2 for both the sales and profit data is specified; therefore, the data for both the independent (X-axis) and dependent (Y-axis) variables are expressed in millions.

Statements 5 and 6 contain the following information:

- Plotting for both sales and profit data is to begin with zero value because FROM 0 has been coded for both XSCATTER and YSCATTER.
- The interval size on the X-axis (sales data) is to be .5 million; the Y-axis interval size is .2 million.
- 24 intervals on the X-axis (out of a maximum of 30) and 18 on the Y-axis (out of a maximum of 20) have been requested.

Statement 7, the REGRESSION command, constitutes the program procedure logic. This command invokes a linear regression analysis with a scatter diagram, as defined in statements 4–6.

To request a linear regression analysis only, you would omit statements 5 and 6 from [Figure 739](#) and only code the statement shown in [Figure 740](#).

```
LINEAR 01 XNAME SALES YNAME PROFIT
```

Figure 740 Linear Regression Analysis Input Statements

Output from VISION:Results

[Figure 741](#) shows an example of the output generated by VISION:Results when a linear regression analysis without a scatter diagram is requested. [Figure 742](#) shows the same with the scatter diagram requested.

11/30/01	VISION:RESULTS REGRESSION ANALYSIS APPLICATION 1	PAGE 1
LABEL OF INDEPENDENT (X) VARIABLE		XVAR
LABEL OF DEPENDENT (Y) VARIABLE		YVAR
REGRESSION LINE		Y = A + BX
WHERE A =		19.9196-
AND B =		3.8720
STANDARD ERROR OF ESTIMATE		14.6635
CORRELATION COEFFICIENT		.7297
COEFFICIENT OF DETERMINATION		.5324
COEFFICIENT OF NON-DETERMINATION		.4676
NO. OF RECORDS PROCESSED		140
TOTAL OF X VALUES		273.0
TOTAL OF Y VALUES		1,731.69-
MINIMUM X VALUE		5.0-
MAXIMUM X VALUE		8.9
MINIMUM Y VALUE		34.44-
MAXIMUM Y VALUE		46.77

Figure 741 Linear Regression Analysis Without Scatter Diagram

VISION:Results automatically calculates the following information:

- Y intercept of the regression line (A).
- Slope of the regression line (B).
- Standard error of the estimate.
- Correlation coefficient.
- Coefficient of determination.
- Coefficient of non-determination.
- Sum of the X values.
- Sum of the Y values.
- Minimum and maximum X values.
- Minimum and maximum Y values.
- Total number of records processed.

11/30/01	VISION:RESULTS REGRESSION ANALYSIS APPLICATION 1	PAGE 1
LABEL OF INDEPENDENT (X) VARIABLE	XVAR	
LABEL OF DEPENDENT (Y) VARIABLE	YVAR	
X STARTING VALUE	5-	
X UNIT VALUE	1	
X NO. OF INTERVALS	16	
Y STARTING VALUE	40-	
Y UNIT VALUE	6	
Y NO. OF INTERVALS	20	
REGRESSION LINE	Y = A + BX	
WHERE A =	19.9196-	
AND B =	3.8720	
STANDARD ERROR OF ESTIMATE	14.6635	
CORRELATION COEFFICIENT	.7297	
COEFFICIENT OF DETERMINATION	.5324	
COEFFICIENT OF NON-DETERMINATION	.4676	
NO. OF RECORDS PROCESSED	140	
TOTAL OF X VALUES	273.0	
TOTAL OF Y VALUES	1,731.69-	
MINIMUM X VALUE	5.0-	
MAXIMUM X VALUE	8.9	
MINIMUM Y VALUE	34.44-	
MAXIMUM Y VALUE	46.77	
NO. OF RECORDS OUTSIDE OF SCATTERGRAM	0	
TOTAL X VALUE OF RECORDS OUTSIDE OF SCATTERGRAM	0.0	
TOTAL Y VALUE OF RECORDS OUTSIDE OF SCATTERGRAM	0.00	

Figure 742 Linear Regression Analysis with Scatter Diagram

When the scatter diagram is requested, the report also includes the following information, in addition to the information previously listed:

- Starting and unit value for X.
- Number of X intervals.
- Starting and unit value for Y.
- Number of Y intervals.
- Number of values that would not fit into the scatter diagram.
- Sum of the X and Y values not included in the scatter diagram.

If a large number of values are not included in the scatter diagram, the program can be rerun with more appropriate starting values, unit values, and number of intervals determined by the minimum and maximum X and Y values.

[Figure 743](#) is an example of the VISION:Results scatter diagram.

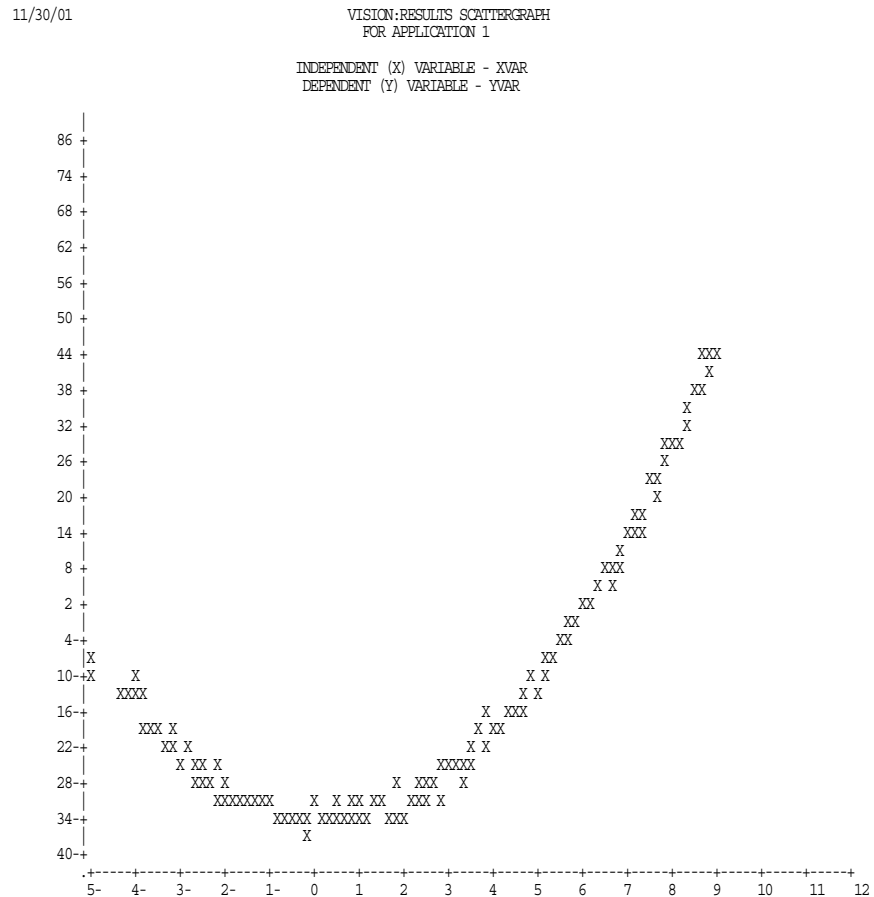


Figure 743 Scatter Diagram

Evaluation of Output

The purpose of linear regression analysis is to find and define a straight line that adequately describes a body of data involving two variables. The coefficient of determination indicates the following:

- Whether or not the two variables are linearly related.
- The form of the data within the scatter diagram.

In the linear regression analysis shown in [Figure 742](#), the coefficient of determination is .5324. Because it is not close to 1, it is very likely that the data does not conform to a straight line very well. The scatter diagram in [Figure 743](#) reinforces this conclusion. Therefore, in this case, the regression line used to estimate the dependent variable from the independent variable would not be very reliable.

Examples of Linear Regression and Trend Line Analysis

Linear Regression Analysis Example

The marketing manager for company ABC would like to determine the relationship between weekly sales and profit for the last 200 weeks. The data necessary for this analysis is maintained on an automated file.

[Figure 744](#) shows the VISION:Results program that performs this analysis.

```
FILE INFILE FB 65 650
SALES 8 1 PD 2
PROFIT 8 PD 2

LINEAR 01 XNAME SALES YNAME PROFIT
XSCATTER 2 FROM 0 BY .5 FOR 24
YSCATTER 2 FROM 0 BY .2 FOR 18

REGRESSION LINEAR 1
```

Figure 744 Linear Regression Analysis Input Statements

[Figure 745](#) is the linear regression analysis.

11/30/01	VISION:RESULTS REGRESSION ANALYSIS APPLICATION 1	PAGE 1
LABEL OF INDEPENDENT (X) VARIABLE	SALES	
LABEL OF DEPENDENT (Y) VARIABLE	PROFIT	
X SCALING FACTOR	MILLIONS	
X STARTING VALUE	0	
X UNIT VALUE	0.5	
X NO. OF INTERVALS	24	
Y SCALING FACTOR	MILLIONS	
Y STARTING VALUE	0	
Y UNIT VALUE	0.2	
Y NO. OF INTERVALS	18	
REGRESSION LINE	$Y = A + BX$	
WHERE A =	22,853.7575	E+1
AND B =	.3016	
STANDARD ERROR OF ESTIMATE	11,800.9206	E+1
CORRELATION COEFFICIENT	.9913	
COEFFICIENT OF DETERMINATION	.9827	
COEFFICIENT OF NON-DETERMINATION	.0173	

Figure 745 Linear Regression Analysis (Page 1 of 2)

NO. OF RECORDS PROCESSED	200
TOTAL OF X VALUES	1,121,897,320.00
TOTAL OF Y VALUES	384,046,724.00
MINIMUM X VALUE	233,910.00
MAXIMUM X VALUE	10,528,290.00
MINIMUM Y VALUE	130,657.00
MAXIMUM Y VALUE	3,509,523.00
NO. OF RECORDS OUTSIDE OF SCATTERGRAM	0
TOTAL X VALUE OF RECORDS OUTSIDE OF SCATTERGRAM	0.00
TOTAL Y VALUE OF RECORDS OUTSIDE OF SCATTERGRAM	0.00

Figure 745 Linear Regression Analysis (Page 2 of 2)

[Figure 746](#) is the scatter diagram.

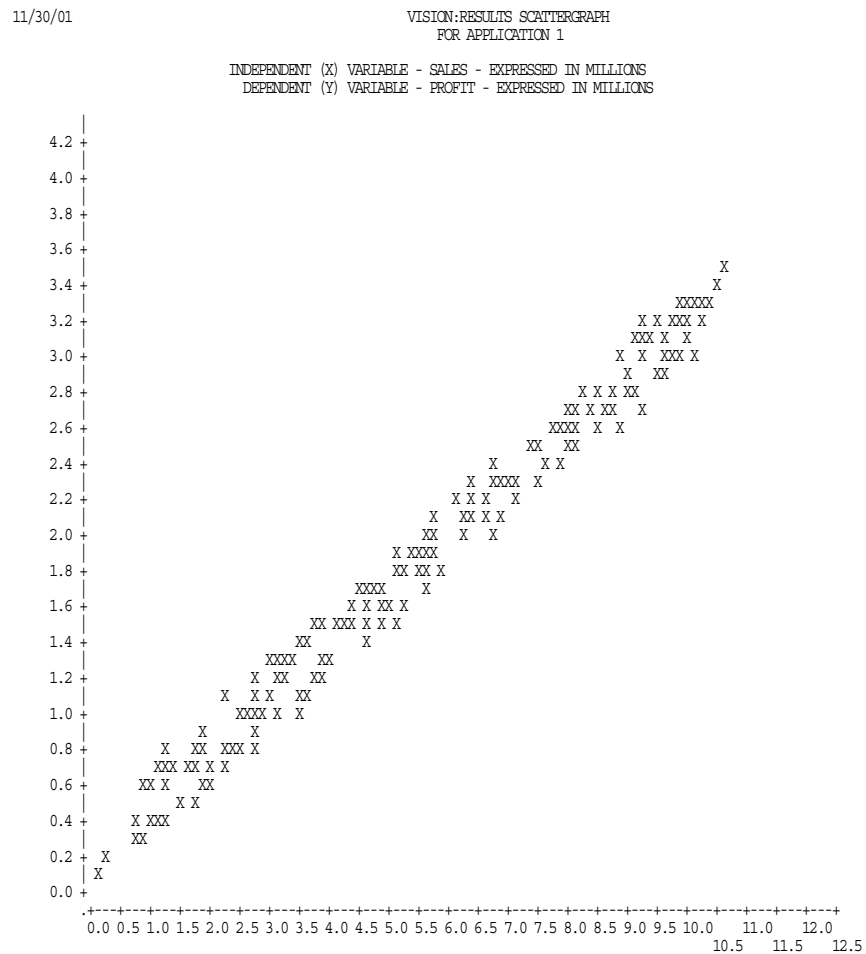


Figure 746 Scatter Diagram

Because the correlation coefficient of .9827 is close to 1, the regression line adequately describes the data. The scatter diagram confirms this conclusion.

Trend Line Analysis Example

The marketing manager for Company XYZ would like to determine the trend in sales over the last 10 years. The sales data is maintained on an automated file.

[Figure 747](#) shows the VISION:Results program requesting a trend line analysis. Note that the title YEARLY SALES TREND, 1988 TO PRESENT prints at the top of both the linear regression analysis and the scatter diagram. This was obtained by including the optional keyword, TITLE, in the trend line statement and following it with a data name. The data name HDRTITLE was chosen and defined with the literal value YEARLY SALES TREND, 1988 TO PRESENT.

```
FILE INFILE INPUT
  YEARS 3 1 PD
  SALES 8 PD 2

WORKAREA
  HDRTITLE 35 1 VALUE 'YEARLY SALES TREND, 1988 TO PRESENT'

TREND 01 XNAME YEARS YNAME SALES TITLE HDRTITLE
  XSCATTER FROM 1988 BY 1 FOR 12
  YSCATTER 1 FROM 0 BY 100 FOR 12

REGRESSION TREND 1
```

Figure 747 Trend Line Analysis Input Statements

[Figure 748](#) is the linear regression analysis.

11/30/01	YEARLY SALES TREND, 1988 TO PRESENT VISION:RESULTS REGRESSION ANALYSIS APPLICATION 1	PAGE 1
LABEL OF INDEPENDENT (X) VARIABLE	YEARS	
LABEL OF DEPENDENT (Y) VARIABLE	SALES	
LABEL OF TITLE FIELD (OPTIONAL)	HDRTITLE	
X STARTING VALUE	1,979	
X UNIT VALUE	1	
X NO. OF INTERVALS	12	
Y SCALING FACTOR	THOUSANDS	
Y STARTING VALUE	0	
Y UNIT VALUE	100	
Y NO. OF INTERVALS	12	
REGRESSION LINE	Y = A + BX	
WHERE A =	22,144.8336-	E+4
AND B =	11,229.5661	E+1
STANDARD ERROR OF ESTIMATE	2,780.9550	E+1
CORRELATION COEFFICIENT		.9963
COEFFICIENT OF DETERMINATION		.9926
COEFFICIENT OF NON-DETERMINATION		.0074
NO. OF RECORDS PROCESSED		10
TOTAL OF X VALUES		19,775
TOTAL OF Y VALUES		6,164,346.11
MINIMUM X VALUE		1,979
MAXIMUM X VALUE		1,989
MINIMUM Y VALUE		74,776.30
MAXIMUM Y VALUE		1,097,729.38
NO. OF RECORDS OUTSIDE OF SCATTERGRAM		0
TOTAL X VALUE OF RECORDS OUTSIDE OF SCATTERGRAM		0
TOTAL Y VALUE OF RECORDS OUTSIDE OF SCATTERGRAM		0.00

Figure 748 Trend Line Linear Regression Analysis

[Figure 749](#) is the scatter diagram.

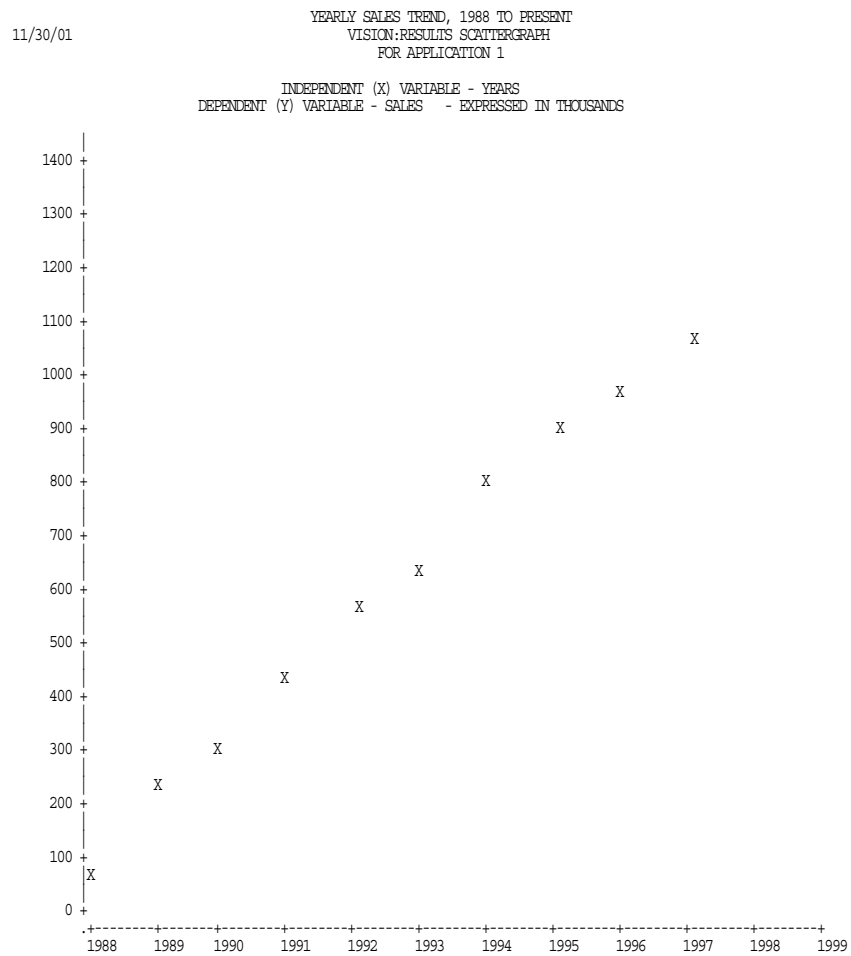


Figure 749 Trend Line Analysis Scatter Diagram

Appendix A: Reserved Words

VISION:Results uses certain words for its own purposes. You should avoid choosing them for your own data names. For example, the following is invalid:

```
FILE ARFILE INPUT FB 100 3000  
COUNT 5 PD
```

COUNT is a VISION:Results reserved word and cannot be used as a data name in a field definition.

All words starting with DYL are reserved, as well as those in the list beginning on the next page.

Note: DYLVARP is a DYLINSTL macro parameter that allows the use of data names prefixed with DYL. It is required only for VISION:Results Interface to DB2 and VISION:Results Interface to SQL/DS. For a definition of DYLVARP, see the *Advantage VISION:Results Interface to DB2 Reference Guide*.

Note: You cannot use the **blue** reserved words (marked with a ♥) as data names if you ever intend to use OPTION EXCEL. You can only use the **red** reserved words (marked with a ♣) and **blue** reserved words (marked with a ♥) as data names if the DYLINSTL parameter RESRWRD is used, but this disables certain VISION:Results capabilities and CA does not recommend doing this.

Symbols

\$\$DUMP
 \$\$KEYLEVEL
 \$\$PCB
 \$COBOL
 \$DEFAULT
 \$DEND
 \$ECOBOL
 \$ELSE ♣
 \$ENDGDEF ♣
 \$ENDGSET ♣
 \$ENDSET ♣
 \$GDEF ♣
 \$GSET ♣
 \$IF
 \$IFE
 \$IFVALUE
 \$PRINT ♣
 \$SET ♣
 \$
 % ♣
 (
)
 *
 **
 +
 -
 /
 @ ♣
 @nnn ♣
 ;
 :
 ,
 =
 % **macroname** ♣
 #anything

A

ABOVE
 ABSOLUTE
 ACCEPT
 ACCEPTABLE ♥
 ACCEPTANCE ♥
 ACCTITLE ♥
 ADD
 ADVANCED ♣
 AFTER
 AGE ♥

AGEDATE ♥
 AGEING ♣
 AGING ♣
 ALIAS
 ALLOCATE
 ALPHANUMERIC ♥
 AMOUNTFIELD ♥
 AND
 AREA ♣
 ARRAY ♣
 ASA
 ASCENDING ♣
 ASCII
 ASIS ♣
 AT
 ATTRIBUTE ♥
 AUDIT ♣

B

BDAM
 BEFORE
 BELOW
 BETWEEN
 BI
 BINSEARCH ♣
 BITS
 BLANK
 BLANKS
 BOOK
 BOTH
 BY
 BYPASS

C

CALL
 CALL_ATTACH
 CARD
 CARDS
 CASE
 CATPLANID
 CATSYSID
 CDLOAD
 CELL ♥
 CH
 CHANGE
 CHnn ♣

CLOSE
 CMX
 COBAPOS
 COBOL
 COBXSIGN
 COB2NR
 COLUMNS
 COMBINE
 COMPAREN ♣
 CON ♣
 CONFIDENCE ♥
 CONTROL
 CONVENTIONAL
 COPY
 COPYC
 COPYDB2
 COPYE
 COPYL
 COPYN
 COPYP
 COPY260
 CORR
 CORRELATION ♥
 CORRESPONDING
 COUNT
 CS
 CSnn ♣
 CTLBRK ♥
 CTRLZ
 CURRENCY
 CYLOFL

D

DA
 DATA
 DATAVER
 DATEFIELD ♥
 DATEFORMAT ♥
 DB2PLANID
 DB2SYSID
 DD/MM/YY ♥
 DD/MM/YYYY ♥
 DDMMYY ♥
 DDMMYYYY ♥
 DDNAME
 DE
 DECLARE
 DELETE ♣
 DELIM
 DESCENDING ♣

Note: You cannot use the **blue** reserved words (marked with a ♥) as data names if you ever intend to use OPTION EXCEL. You can only use the **red** reserved words (marked with a ♣) and **blue** reserved words (marked with a ♥) as data names if the DYLINSTL parameter RESRWRD is used, but this disables certain VISION:Results capabilities and CA does not recommend doing this.

DETAIL ♥

DIM ♣

DIMENSION ♣

DIRECTORY

DISCOVERY ♥

DISK

DISKETTE

DISKPASSWORD

DISTFIELD ♥

DLM

DMAP

DNA

DNE

DOLLAR ♥

DOS

DOUNTIL

DOWHILE

DROPCRLF

DROPERR

DUMMY

DUPLICATE ♣

DYACCEPT

DYL

DYL-SP

DYLCNAME0A

DYLCNAME0B

DYLCNAME0C

DYLCNAME0D

DYLCNAME0E

DYLCNAME0F

DYLCNAME0G

DYLCNAME0H

DYLCNAME0I

DYLCNAME0J

DYLCNAME0K

DYLCNAME0L

DYLCNAME0M

DYLCNAME0N

DYLCNAME0O

DYLCNAME0P

DYLCNAME0Q

DYLCNAME0R

DYLCNAME0S

DYLCNAME0T

DYLCNAME0U

DYLCNAME0V

DYLCNAME0W

DYLCNAME0X

DYLCNAME0Y

DYLCNAME0Z

DYLCNAME00

DYLCNAME1

DYLCNAME10

DYLCNAME11

DYLCNAME12

DYLCNAME13

DYLCNAME14

DYLCNAME15

DYLCNAME16

DYLCNAME17

DYLCNAME18

DYLCNAME19

DYLCNAME2

DYLCNAME20

DYLCNAME21

DYLCNAME22

DYLCNAME23

DYLCNAME24

DYLCNAME25

DYLCNAME26

DYLCNAME27

DYLCNAME28

DYLCNAME29

DYLCNAME3

DYLCNAME30

DYLCNAME31

DYLCNAME32

DYLCNAME33

DYLCNAME34

DYLCNAME35

DYLCNAME36

DYLCNAME37

DYLCNAME38

DYLCNAME39

DYLCNAME4

DYLCNAME40

DYLCNAME41

DYLCNAME42

DYLCNAME43

DYLCNAME44

DYLCNAME45

DYLCNAME46

DYLCNAME47

DYLCNAME48

DYLCNAME49

DYLCNAME5

DYLCNAME50

DYLCNAME51

DYLCNAME52

DYLCNAME53

DYLCNAME54

DYLCNAME55

DYLCNAME56

DYLCNAME57

DYLCNAME58

DYLCNAME59

DYLCNAME6

DYLCNAME60

DYLCNAME61

DYLCNAME62

DYLCNAME63

DYLCNAME64

DYLCNAME65

DYLCNAME66

DYLCNAME67

DYLCNAME68

DYLCNAME69

DYLCNAME7

DYLCNAME70

DYLCNAME71

DYLCNAME72

DYLCNAME73

DYLCNAME74

DYLCNAME75

DYLCNAME76

DYLCNAME77

DYLCNAME78

DYLCNAME79

DYLCNAME8

DYLCNAME80

DYLCNAME81

DYLCNAME82

DYLCNAME83

DYLCNAME84

DYLCNAME85

DYLCNAME86

DYLCNAME87

DYLCNAME88

DYLCNAME89

DYLCNAME9

DYLCNAME90

DYLCNAME91

DYLCNAME92

DYLCNAME93

DYLCNAME94

DYLCNAME95

DYLCNAME96

DYLCNAME97

DYLCNAME98

DYLCNAME99

DYLCCOMMON

DYLCOMRG

DYLCOUNT0A

DYLCOUNT0B

DYLCOUNT0C

DYLCOUNT0D

DYLCOUNT0E

DYLCOUNT0F

DYLCOUNT0G

DYLCOUNT0H

DYLCOUNT0I

DYLCOUNT0J

Note: You cannot use the **blue** reserved words (marked with a ♥) as data names if you ever intend to use OPTION EXCEL. You can only use the **red** reserved words (marked with a ♣) and **blue** reserved words (marked with a ♥) as data names if the DYINSTL parameter RESRWRD is used, but this disables certain VISION:Results capabilities and CA does not recommend doing this.

DYLCOUNT0K
DYLCOUNT0L
DYLCOUNT0M
DYLCOUNT0N
DYLCOUNT0O
DYLCOUNT0P
DYLCOUNT0Q
DYLCOUNT0R
DYLCOUNT0S
DYLCOUNT0T
DYLCOUNT0U
DYLCOUNT0V
DYLCOUNT0W
DYLCOUNT0X
DYLCOUNT0Y
DYLCOUNT0Z
DYLCOUNT00
DYLCOUNT1
DYLCOUNT10
DYLCOUNT11
DYLCOUNT12
DYLCOUNT13
DYLCOUNT14
DYLCOUNT15
DYLCOUNT16
DYLCOUNT17
DYLCOUNT18
DYLCOUNT19
DYLCOUNT2
DYLCOUNT20
DYLCOUNT21
DYLCOUNT22
DYLCOUNT23
DYLCOUNT24
DYLCOUNT25
DYLCOUNT26
DYLCOUNT27
DYLCOUNT28
DYLCOUNT29
DYLCOUNT3
DYLCOUNT30
DYLCOUNT31
DYLCOUNT32
DYLCOUNT33
DYLCOUNT34
DYLCOUNT35
DYLCOUNT36
DYLCOUNT37
DYLCOUNT38
DYLCOUNT39
DYLCOUNT4
DYLCOUNT40
DYLCOUNT41
DYLCOUNT42

DYLCOUNT43
DYLCOUNT44
DYLCOUNT45
DYLCOUNT46
DYLCOUNT47
DYLCOUNT48
DYLCOUNT49
DYLCOUNT5
DYLCOUNT50
DYLCOUNT51
DYLCOUNT52
DYLCOUNT53
DYLCOUNT54
DYLCOUNT55
DYLCOUNT56
DYLCOUNT57
DYLCOUNT58
DYLCOUNT59
DYLCOUNT6
DYLCOUNT60
DYLCOUNT61
DYLCOUNT62
DYLCOUNT63
DYLCOUNT64
DYLCOUNT65
DYLCOUNT66
DYLCOUNT67
DYLCOUNT68
DYLCOUNT69
DYLCOUNT7
DYLCOUNT70
DYLCOUNT71
DYLCOUNT72
DYLCOUNT73
DYLCOUNT74
DYLCOUNT75
DYLCOUNT76
DYLCOUNT77
DYLCOUNT78
DYLCOUNT79
DYLCOUNT8
DYLCOUNT80
DYLCOUNT81
DYLCOUNT82
DYLCOUNT83
DYLCOUNT84
DYLCOUNT85
DYLCOUNT86
DYLCOUNT87
DYLCOUNT88
DYLCOUNT89
DYLCOUNT9
DYLCOUNT90
DYLCOUNT91

DYLCOUNT92
DYLCOUNT93
DYLCOUNT94
DYLCOUNT95
DYLCOUNT96
DYLCOUNT97
DYLCOUNT98
DYLCOUNT99
DYLDATE
DYLDATE4
DYLDATEPAG
DYLDATEPG4
DYDLA
DYLETIME
DYLEXTA
DYLEXTAA
DYLFQX
DYLFQxx
DYLGREG
DYLGREG4
DYLILDATE
DYJULIAN
DYJULIAN4
DYLLINE
DYLNOTOT
DYLNRP
DYLONG
DYLPAGE
DYLPAGE1
DYLPAGE2
DYLPAGE3
DYLPAGE4
DYLPAGE5
DYLPAGE6
DYLPAGE7
DYLPARM
DYLPARMLEN
DYLPAGE
DYLPRA
DYLPORGRAM
DYLPRTCOMM
DYLPRTMAXL
DYLPRTNUMB
DYLPRTPAGE
DYLRMAIN
DYLRPMAXL
DYLRSET
DYLRTURN
DYLSYSDATE
DYLSYSPARM
DYLTIME
DYLTAA
DYLTAA
DYLTRAN

Note: You cannot use the **blue** reserved words (marked with a ♥) as data names if you ever intend to use OPTION EXCEL. You can only use the **red** reserved words (marked with a ♣) and **blue** reserved words (marked with a ♥) as data names if the DYLINSTL parameter RESRWRD is used, but this disables certain VISION:Results capabilities and CA does not recommend doing this.

DYLUPSI
DYL280RA
DYL4YEAR
DYNAM
DYNAMDB2
DYNEXT
DYSTORE
D4

E

EBCDIC
EDIT
EDOS
EJECT ♣
ELSE
END
ENDCASE
ENDDO
ENDIF
ENDLTD ♣
ENDONE
ENTRY ♣
EOS
EQ
EQUAL ♥
ERASE
ERROR
ESDS
ESTIMATION ♥
EXCEL
EXCELPAT
EXCLUSIVE
EXEC
EXIT
EXITCASE
EXITDO
EXOR
EXPTERR
EXT
EXTENSION
EXTENTS

F

F
FB
FBA
FFFF
FFFFEE
FFFFFF
FIELDNAME
FIELDS
FILE
FILENAME
FILES
FILL ♣
FILLER
FINAL
FIRSTDUP ♣
FIXED
FMX
FOOTING
FOR ♣
FORMAT
FREE
FREEZDD
FREEZE
FREQ ♥
FREQUENCY
FROM
FUTURE ♥

G

GE
GETCOM
GOTO
GROUP1 ♥
GROUP10 ♥
GROUP2 ♥
GROUP3 ♥
GROUP4 ♥
GROUP5 ♥
GROUP6 ♥
GROUP7 ♥
GROUP8 ♥
GROUP9 ♥
GT

H

HEX
HEXPRINT
HIDDEN
HIGHAMT ♥
HIGHVALUES
HISTOGRAM ♥
HOLD
HTAB
HTML
HTMLTITL

I

IDLEN ♣
IDLENGTH ♣
IDMSMOVE
IF
INDENT ♣
INPUT
INTERPRET
INTERVAL ♥
INTERVALNO ♥
INTERVALSIZE ♥
INTO
INTSEL
INW
INX
INY
INZ
IO
IQBATCH
IQFILE
IQREAD
ISAM
ITEMFIELD ♥
ITEMNO ♥
ITERATE

J

JCL

Note: You cannot use the **blue** reserved words (marked with a ♥) as data names if you ever intend to use OPTION EXCEL. You can only use the **red** reserved words (marked with a ♣) and **blue** reserved words (marked with a ♥) as data names if the DYLINSTL parameter RESRWRD is used, but this disables certain VISION:Results capabilities and CA does not recommend doing this.

K

KEY ♥
KEYLEN
KEYLOC
KEYn ♣
KSDS

L

LASERnn ♣
LASTDUP ♣
LCPRINT
LE
LEFT
LEFTSAVE ♣
LENGTH
LETTER ♣
LIB ♣
LIBNAME
LIBPASSWORD
LINEAR ♥
LIST
LISTAG
LISTAUD
LISTINS
LIST26
LOGARITHMIC ♥
LOGIC
LONG
LOWAMT ♥
LOWER
LOWVALUES
LT
LTD ♣
LTDnn ♣
LTH ♣
L2U

M

MASTER
MASTERUP ♣
MATCH ♣
MATCHED ♣
MAXBLKSIZE
MAXIMUM ♥

MEAN ♥
MEMBER
MERGE ♣
MIXED
MM/DD/YY ♥
MM/DD/YYYY ♥
MMDDYY ♥
MMDDYYYY ♥
MODIFY
MONETARY ♥
MOVE
MQBACK
MQBEGIN
MQCLOSE
MQCMIT
MQCONN
MQCONNx
MQDISC
MQGET
MQINQ
MQOPEN
MQPUT
MQPUT1
MQSET
MSGCSECT

N

NA
NAMED
NB
NDA
NDE
NDV
NDVRCOM
NDVRENV
NDVRSTG
NDVRSUB
NDVRSYS
NDVRTYP
NE
NEGATIVE
NEW
NEWEXT
NEWEXTENSION
NEWFILENAME
NEWNAME
NEWPAGE
NEWPARA ♣
NEXT
NL

nn
NO ♥
NOCLOAD
NOCOB2NR
NOCOBAPOS
NOCOBXSIGN
NODYL4YEAR
NOEDIT ♣
NOEJECT ♣
NOERROR
NOEXCEL
NOEXCELPAT
NOEXPRTERR
NOFEED
NOLE
NONDVRCOM
NONUMCHAR
NONUMPD
NOOPTIMIZE ♣
NOOPTLIST
NOPRINTCTRS
NOPRINTD
NOPRINTDIGITS
NOPRINTEP
NOSYSBLOCK
NO_PRINT_ENTRY_POI
NTS
NOQLF
NOQUALIFIERS
NORETAIN
NORMALIZED
NORWD
NOSORTAB
NOSPOOL
NOSQL2DBC
NOSUM
NOT
NOTOTAL
NOVSAMCAT
NOVSEALL
NOVSEDISK
NOVSETAPE
NOXREF
NP
NU
NULL
NULLOFF
NULLON
NUMCHAR
NUMERIC
NUMFILES
NUPD
NZ

Note: You cannot use the **blue** reserved words (marked with a ♥) as data names if you ever intend to use OPTION EXCEL. You can only use the **red** reserved words (marked with a ♣) and **blue** reserved words (marked with a ♥) as data names if the DYLINSTL parameter RESRWRD is used, but this disables certain VISION:Results capabilities and CA does not recommend doing this.

O

OFFSET
ON
ONE
ONEBUFF
ONES
OPTIMIZE ♣
OPTION
OPTIONS
OPTLIST
OR
ORIGINAL
OS
OTHERWISE
OUTPUT
OVERAUDITING ♥

P

PAD
PAN ♣
PARM
PARTKEY
PASSWORD
PAST ♥
PATCHES
PATHNAME
PATHNAME01
PATHNAME02
PATHNAME03
PATHNAME04
PATHNAME05
PATHNAME06
PATHNAME07
PATHNAME08
PATHNAME09
PATHNAME1
PATHNAME10
PATHNAME11
PATHNAME2
PATHNAME3
PATHNAME4
PATHNAME5
PATHNAME6
PATHNAME7
PATHNAME8
PATHNAME9
PCFILE
PCWRITE

PD
PDS
PDSREPA
PDSREPN
PDO
PERCENTAGE ♥
PERFORM
PICNSAVE
PLANID
PLUnnn
PMX
POPULATION ♥
POSITION
POSITIVE
PRECISION
PRINT
PRINTCTRS
PRINTD
PRINTDIGITS
PRINTEP
PRINT_ENTRY_
POINTS
PRINTERR
PRINTERROR
PRINTGEN ♣
PRINTNOGEN ♣
PRIOR
PROPORTIONAL ♥
PROTECT
PUTCOM

Q

QLF
QUALIFIERS
QUIKJOB
QUIT
QUITALL

R

RANDOM
RANDOMX
RANDPROG ♥
RANDPROG SIZE ♥
READ
READBOOK
READDIR

READIR
READMEM
READONLY
RECFM
REDEF
REDEFINE
REGRESSION ♣
REINIT
REJECT
RELBLK
RELBYTE
RELBYTEX
RELIABILITY ♥
RELREC
RELSORT
RELTRK
REPLACE
REPLACING
REPLICA ♣
REPORT
REPORTFILE
REPORTn
REPORTnn
REPORTnnn
REPTITLE
RESTART
RESTORE
RETAIN
RETRIEVE ♣
RETSORT
REUSE
REWIND
RIGHT
ROUND
ROUNDED
RRDS

S

SAMPLE ♥
SAMPLING
SB
SCATTER ♣
SCHEMADD
SCHEMADS
SCHEMAHF
SCHEMAUL
SCRATCH
SEARCH ♣
SECURE
SEED ♥

Note: You cannot use the **blue** reserved words (marked with a ♥) as data names if you ever intend to use OPTION EXCEL. You can only use the **red** reserved words (marked with a ♣) and **blue** reserved words (marked with a ♥) as data names if the DYLINSTL parameter RESRWRD is used, but this disables certain VISION:Results capabilities and CA does not recommend doing this.

SELECT
 SEQUENTIAL
SETREAD ♣
 SIZE
 SKIP
 SKIPX
 SL
SMALL ♥
 SORT
 SORTING
 SPACE
 SPACES
SPn ♣
 SPT
 SPOOL
 SQL2DBC
 SSL
STANDARD ♥
START ♥
 STATEOFF
 STATEON
 STATSQ
 STATUS
 STATUSFLAG
STDREPT ♥
 STOP
 STOPALL
STORE ♣
STRATIFIED ♥
STRATUM1 ♥
STRATUM10 ♥
STRATUM2 ♥
STRATUM3 ♥
STRATUM4 ♥
STRATUM5 ♥
STRATUM6 ♥
STRATUM7 ♥
STRATUM8 ♥
STRATUM9 ♥
 STRUCTURED
 STRUCTURED2
 STYLE
 SUBDIR
 SUBTOTAL
 SUM
SUMMARY ♥
 SUM1
 SUM2
 SUM3
 SUM4
 SUM5
 SUM6
 SUM7
 SUPPRESS

SWITCH ♣
 SYR
 SYSBLOCK
 SYSnnn
 SYSTEMID
 SYS280RA
 S2R

T

TABAREA
 TALLYn
TABLE ♣
 TAPE
 TEST
 TEXT
 THEN
 THROUGH
 THRU
 TIMECOLON
 TIMEDOT
TITLE ♣
 Tn
 TO
 TOTAL
 TPRINT
 TRAN
 TRANSLATE
TREND ♣
TSIZE ♣
 TSO_ATTACH
TSORT ♣
TWOP ♣

U

UDTL
UNIT ♥
 UNL
 UNLOAD
 UNSIGNED
 UNTIL
UNWARRANTED ♥
 UPPER
 USE

USERDEFAULT
 USERINX
 USERINXnn
 USING

V

V
 VAL
 VALUE
 VARYING
 VB
 VERIFY
 VFILEON
 VOLVER
 VSAM
 VSAMCAT
 VSAMEC
 VSAMRC
 VSEALL
 VSEDISK
 VSETAPE

W

WHEN
 WIDE
 WITH
 WORK
 WORKAREA
WORKSHEET ♥
 WRITE
WRITEALL ♣
 WRITEDIR
 WRITEMEM

Note: You cannot use the **blue** reserved words (marked with a ♥) as data names if you ever intend to use OPTION EXCEL. You can only use the **red** reserved words (marked with a ♣) and **blue** reserved words (marked with a ♥) as data names if the DYLINSTL parameter RESRWRD is used, but this disables certain VISION:Results capabilities and CA does not recommend doing this.

X

XLATE
XMLFILE
XMLWRITE
XNAME ♣
XREF
XREFA
XSCATTER ♣

Y

YES ♥
YNAME ♣
YR1
YR2
YSCATTER ♣
YY/MM/DD ♥
YYDDD ♥

YYMMDD ♥**YYYY/MM/DD** ♥**YYYYMMDD** ♥**YYYYDDD** ♥

Y2C

Y2D

Y2P

Y2S

Y2B

Y2Z

Z

ZDIVAB
ZDIVORG
ZDIVRC
ZERO ♥
ZONE

Numbers

2311

2314

260

3330

3340

3350

3375

3380

Definitions

Name Type	Length	Data Type	Edit	Decimals
BLANK or BLANKS	1	CH		
One space. Can be used to space fill any size character field because a character field is space-filled to the right after a value is moved into it.				
DYLCENTRY1 DYLCENTRY2	2	CH		
Use to assign a century prefix for 2-digit years to produce a 4-digit output. If DYLCENTRY1 or DYLCENTRY2 parameters are not set, VISION:Results defaults to the values set in the DYLINSTL parameters CENTRY1 and CENTRY2, respectively. If no values are set in CENTRY1 and CENTRY2, the system defaults to the DYLINSTL parameter CENTNEW.				
DYLCNAMEnn	20	CH		
Contains 126 twenty-byte label fields that you can set to identify the corresponding DYLCOUNTnn counters when they are printed. nn may be 1-99, 00, and 0A-0Z. If you do not move your own labels into these fields, the standard identifiers are to the left of the counters as TOTAL NUMBER 01, TOTAL NUMBER 02, and so on.				

Name Type	Length	Data Type	Edit	Decimals
DYLCOMMON	4	BI		
This field contains the address of the VISION:Results common area. It is used as a parameter that is passed using CALL to a user module. You might need this to gain access to the VISION:Results DTF/DCBs.				
DYLCOMRG	11	CH		
The value is used for multiple purposes. It is a copy of bytes 12-22 of the VSE partition communications region. It is used to communicate the XML API completion code and reason code. It contains the value of register 15 for the CSV routines, TOCSV and FROMCSV. Changing the value in DYLCOMRG has no effect on the actual value in the communications region.				
DYLCOUNTnn	10	PD	A	0
Contains 126 ten-byte packed counters that you can use to maintain your own totals. nn may be 1-99, 00, and 0A-0Z. If you reference any of these counters, the series is printed as part of the end-of-run file statistics.				
DYLDATE	8	CH		
The initial value is the current date in MM/DD/YY format.				
DYLDATE4	10	CH		
The initial value is the current date in MM/DD/YYYY format.				
DYLDATEPAG	30	CH		
The current date and page number edited and ready for insertion in report titles. The format is:				
DATE MM/DD/YY PAGE NNNNNNN				
DYLDATEPG4	32	CH		
The current date and page number edited and ready for insertion in report titles. The format is:				
DATE MM/DD/YYYY PAGE NNNNNNN				
DYLDLA	4	BI		
The first parameter in a CALL statement when invoking DYLTDLI (IMS/DLI database interface option). DYLDLA should never be modified in your program.				
DYLETIME	8	CH		
Value is the execution start time of VISION:Results in HH.MM.SS format.				
DYLGREG	4	PD	D	
The current Gregorian date in packed format. Packed format is 0MMDDYYs.				

Name	Type	Length	Data Type	Edit	Decimals
DYLGREG4		5	PD	D	
The current Gregorian date in packed format. Packed format is 0MMDDYYYYs.					
DYLIIDATE		4	BI		
The current Lilian date in binary format.					
DYLGJULIAN		3	PD		
The current Julian date in packed format. The format is YYDDD.					
DYLGJULIAN4		4	PD		
The current Julian date in packed format. The format is YYYYDDD.					
DYLLINE		2	PD	A	
The line count of the current report page in packed format.					
DYLNOTOT		1	CH		
Control total inhibit. Can be set by the user.					
Ø = Not used (initial setting).					
1 = Do not increment control break totals on this entrance to the report writer.					
DYLNRP		1	CH		
A detail line inhibit switch that you can set (enter a 2) before the first detail line LIST statement is issued.					
DYLONG		1	PD		
Value 1. Useful for counting.					
DYLPAGE		13	CH		
The literal PAGE followed by the edited page number. The format is: PAGE NNNNNNNN					
DYLPAGE1		1	CH		
Edited 1-character page number.					
DYLPAGE2		2	CH		
Edited 2-character page number.					
DYLPAGE3		3	CH		
Edited 3-character page number.					
DYLPAGE4		4	CH		
Edited 4-character page number.					

Name	Type	Length	Data Type	Edit	Decimals
DYLPAGE5		5	CH		
Edited 5-character page number.					
DYLPAGE6		6	CH		
Edited 6-character page number.					
DYLPAGE7		7	CH		
Edited 7-character page number.					
DYLPARM		60	CH		
Variable data provided by either the PARM parameter in the EXEC statement (z/OS only) or the DATA xxxx phrase in the OPTION statement.					
DYLPARMLEN		2	PD		
The length of the data stored in DYLPARM.					
DYLPPAGE		6	PD	A	
The current report page number. Can be reset.					
DYLPROGRAM		8	CH		
z/OS: A user-specified program identification name. The default is the library member name assigned to the SYSIN DD statement.					
DYLPRTCOMM		10	CH		
This field can be set to hold an identifier to print to the left of a record or field being printed using the print (PRINT, HEXPRINT, and so on) commands. If you do not move anything in before printing, the file name or data name appears.					
DYLPRTMAXL		2	PD		
The maximum lines per page to be allowed for file printing. It is set either by default or by using the LONG keyword of the OPTION statement.					
DYLPRTNUMB		4	PD		
The count of records file printed by VISION:Results.					
DYLPRTPAGE		3	PD		
The page number for the file print listing.					
DYLREPMAXL		2	PD		
The maximum lines per page allowed for the report.					
DYLRESET		1	BI		
Reset report page number after break indicator.					
0	=	Not used (initial setting).			
1-6	=	Reset report page number to 1 after break 1-6.			

Name	Type	Length	Data Type	Edit	Decimals
DYLRETURN		2	BI		
Set this before a run terminates to cause a condition code to be passed to the next step or to cause an ABEND to occur. Setting DYLRETURN does not cause the run to terminate. Processing continues until all files are at end, or a STOP or a QUIT command is executed.					
0	=	Normal completion.			
1-99	=	Condition code value to be passed.			
100-4095	=	ABEND with system dump when processing terminates.			
DYLSYSDATE		9	NU		
VSE only. The system date from the partition communication region (offset X`4F'). The first six bytes contain the Gregorian date (MMDDYY or DDMMYY), and the last three bytes contain the Julian day number (DDD). This date is not affected by the // DATE statement.					
DYLSYSPARM					
z/OS: DYLSYSPARM contains the PCB address list reflected in the PSB. For details, see Database Access in the <i>Advantage VISION:Results for z/OS Reference Guide</i> .					
VSE: Size 8, type CH. DYLSYSPARM contains 0 to 8 bytes of data from the SYSPARM=statement on your EXEC statement, padded on the right with blanks. This is release-dependent.					
DYLTIME		4	PD	A	
Value is the execution start time of VISION:Results. Packed format is 0HHMMSS.					
DYLTRAN		1	CH		
Refers to the first byte of a 256-byte translate table. If you do not want to use the FROM and TO option of the TRANSLATE command, you can specify your own translate table by moving values to the appropriate offset from DYLTRAN.					
DYLUPSI		1	BI		
Value is a copy of the VSE UPSI byte. Changing this value has no effect on the actual UPSI byte.					
INW		2	BI	A	
An index used to manipulate a table or fields and loop through records.					
INX		2	BI	A	
An index used to manipulate a table or fields and loop through records.					
INY		2	BI	A	
An index used to manipulate a table or fields and loop through records.					
INZ		2	BI	A	
An index used to manipulate a table or fields and loop through records.					

Name Type	Length	Data Type	Edit	Decimals
SPACE or SPACES	1	CH		
One space. Can be used to space fill any size character field because a character field is space-filled to the right after a value is moved into it. Same as BLANK or BLANKS.				
TALLY	16	PD	Z	9
The total of records printed or accepted in the report since the last control break.				

Appendix B: Job Control Language

This appendix provides examples of the JCL for VISION:Results z/OS, VSE, and VM/CMS.

z/OS JCL

```
//RESULTS JOB .....accounting information
//STEP01 EXEC PGM=DYL280,REGION=600K
//STEPLIB DD DSN=your.results.loadlib,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYS280R DD SYSOUT=A
//SYS004 DD UNIT=3380,SPACE=(TRK,5)
//xxxxxxx DD DSN=your.input.file,DISP=SHR
//SYSIN DD *

your VISION:Results statements
/*
//
```

If the SORT command is coded in your program, you also need:

```
//SYSOUT DD SYSOUT=A
//SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
//SORTWK01 DD UNIT=3380,SPACE=(CYL,5),,CONTIG)
//SORTWK02 DD UNIT=3380,SPACE=(CYL,5),,CONTIG)
//SORTWK03 DD UNIT=3380,SPACE=(CYL,5),,CONTIG)
```

If the COPY command is coded in your program, you also need the following to tell VISION:Results what library contains the members.

```
//SYSCOPY DD DSN=your.copylib,DISP=SHR
```

VSE JCL

```
// JOB RESULTS

1 // ASSGN SYS004,X'cuu'
// DLBL IJSYS04,'VISION:Results work file',0
// EXTENT SYS004,.....,.....

2 // ASSGN SYSxxx,X'nnn'
// DLBL filename,'your.input file'
// EXTENT SYSxxx,.....,extent information
```

```
// EXEC DYL280,SIZE=600K

                                your VISION:Results statements

/*
/ &
```

1. Assign only if IJSYS04 is not standard label.
2. Defines the input file JCL if disk (see the *Advantage VISION:Results for z/OS Getting Started*).

If the SORT command is used in your program, you also need:

```
// DLBL   SORTWK1,'results sort work 1',0
// EXTENT SYS001,...extent information
// DLBL   SORTWK2,'results sort work 2',0
// EXTENT SYS002,...extent information
// DLBL   SORTWK3,'results sort work 3',0
// EXTENT SYS003,...extent information
```

} Only if WORK N keyword
is coded on the SORT
statement

VM/CMS

File definition statements:

```
ERASE SYSPRINT LISTING
ERASE SYS280R LISTING
ERASE SYS004 DATA
FILEDEF SYSPRINT DISK SYSPRINT LISTING A (DISP MOD)
FILEDEF SYS280R DISK SYS280R LISTING A
FILEDEF SYS004 DISK SYS004 DATA A
1 FILEDEF EMPMSTR DISK DYLDATA DATA A
2 FILEDEF SYSIN DISK DYLPROG DATA A
DYL280
```

1. Informs CMS that a disk file with the file name DYLDATA and the file type DATA is referred to as EMPMSTR in your VISION:Results program. In this example, this would be your input file.
2. Informs CMS that a disk file with the file name DYLPROG and the file type DATA is referred to as SYSIN by VISION:Results. This file contains your program statements.

Appendix C: z/OS Utilities

This appendix describes the following VISION:Results utilities:

- Copy sequential files to sequential files (fixed, variable, variable spanned, and undefined formats).
- Copy files with selection.
- Print records or fields from sequential, [ISAM](#), or VSAM files.
- Create [ISAM](#) files from sequential, ISAM, or VSAM files.
- Backup [ISAM](#) files from sequential files.
- Reorganize [ISAM](#) files.
- Add members to partitioned data sets.
- Create and restore VSAM KSDS and ESDS files.
- Backup VSAM KSDS and ESDS files.

The examples presented here by no means exhaust the possible uses of VISION:Results. They are intended to provide a base that you can expand upon. The full range or depth of VISION:Results as an extended utility can be realized only through its use and application.

JCL Considerations

The basic VISION:Results JCL requirements are:

//jobname	JOB	
//JOB LIB	DD	Can be required to identify the VISION:Results load library.
//stepno	EXEC PGM=DYL280	
//STEPLIB	DD	Can be required to identify the VISION:Results load library.
//SYS PRINT	DD SYSOUT=A	Holds source listing, totals, file print, and error analysis.
//SYS004	DD UNIT=SYSDA,	VISION:Results work file.
//	SPACE=(TRK,(10,1),RLSE)	

<code>//filename</code>	<code>DD</code>	Defines the files you are processing in your program. The ddname must be the same as the file name in your VISION:Results FILE statement.
<code>//SYSIN</code>	<code>DD</code>	Defines the file that contains the VISION:Results program statements. Can also contain data if you specify instream data in one of your VISION:Results FILE statements.

If you have LIST statements in your VISION:Results program to produce a report, you also need:

```
//SYS280R      DD      SYSOUT=A
```

If you have COPY statements in your VISION:Results program to include file definitions or other pre-coded source statements in the program stream, you need:

```
//SYSCOPY      DD      Defines source PDS containing VISION:Results members.
```

If you issue a SORT statement in your VISION:Results program, you need:

<code>//SYSOUT</code>	<code>DD SYSOUT=A</code>	Message file for SORT.
<code>//SORTLIB</code>	<code>DD DSN=SYS1.SORTLIB,DISP=SHR</code>	SORT routine library.
<code>//SORTWK01</code>	<code>DD UNIT=SYSDA,SPACE=</code>	SORT work file 1.
<code>.</code>		
<code>.</code>		
<code>//SORTWK0n</code>	<code>DD UNIT=SYSDA,SPACE=</code>	SORT work file n.

Input File Definitions

If the input file you are processing is standard labeled and you are not sorting the file in your program, your FILE statement usually has to contain only the file name and the file organization (physical sequential is the default if omitted). VSAM file definitions always need the record format (F, V, or S). RRDS is the exception, which allows only F. File definitions are not needed, however, if VSAMCAT has been specified in the DYLINSTL macro or the OPTION statement.

<code>FILE FILEIN</code>	Sequential organization.
<code>FILE FILEIN KSDS F</code>	VSAM KSDS fixed format file.
<code>FILE FILEIN KSDS V</code>	VSAM KSDS variable format file.
<code>FILE FILEIN KSDS S</code>	VSAM KSDS variable spanned format file.
<code>FILE FILEIN ESDS F</code>	VSAM ESDS fixed format file.

FILE FILEIN ESDS V	VSAM ESDS variable format file.
FILE FILEIN ESDS S	VSAM ESDS variable spanned format file.
FILE FILEIN RRDS F	VSAM RRDS variable spanned format file.
FILE FILEIN ISAM	ISAM fixed or variable format file.

If you are sorting a standard labeled input file in your VISION:Results program, you need to include record format and record size information in your FILE statement.

```
FILE FILEIN FB 80
  SORTFIELD 5 3
SORT FILEIN USING SORTFIELD

FILE FILEIN KSDS F 120
  SORTFIELD 10 31
SORT FILEIN USING SORTFIELD
```

Figure 750 FILE Statement Example

If your input file is unlabeled, in addition to file name and organization you need to provide information about record format, length, and block size. This can be done either in the DCB parameters of the file's JCL or in the FILE statement of the VISION:Results program.

```
//FILEIN DD UNIT=TAPE,VOL=SER=123456,DISP=OLD,LABEL=(,NL),
//          DSN=MYFILE,DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600)
.
.
FILE FILEIN

or

//FILEIN DD UNIT=TAPE,VOL=SER=123456,DISP=OLD,LABEL=(,NL),
//          DSN=MYFILE
.
.
FILE FILEIN FB 80 1600
```

If you are sorting an unlabeled file in your VISION:Results program, you must include the record format and size information in the FILE statement, even though it can also be in the DCB parameter of the JCL.

```
FILE FILEIN FB 80
  SORTFIELD 5 3
SORT FILEIN USING SORTFIELD
```

Figure 751 Record Format and Size Information Example

[ISAM](#) or VSAM files that are accessed in random or skip sequential mode require that additional items be coded (access mode, key length, status).

Output File Definitions

Output file definitions require file attributes in either the JCL or the FILE statement (as in the case of a VSAM file) unless the file is preallocated or predefined. File organization, if not physical sequential, must always be coded in the FILE statement ([ISAM](#), KSDS, ESDS, RRDS). The area from where the file is being written must always be specified (example, FROM FILEIN).

```
//FILEOUT DD DSN=OUTFILE,UNIT=SYSDA,VOL=SER=DISK01,
//          SPACE=(TRK,(20,1),RLSE),DISP=(,CATLG,DELETE),
//          DCB=(RECFM=FB,LRECL=120,BLKSIZE=1200)
.
.
FILE FILEOUT OUTPUT FROM FILEIN
```

If you are building the record in the output area, you must specify the output file's format and length.

```
FILE FILEOUT OUTPUT FROM FILEOUT FB 120
  RECOUT 120
```

Figure 752 Output File's Format and Length Example

Utilities

This section describes the utilities provided by VISION:Results.

Standard Label Tape to Standard Label Tape

Normal

Because the input tape is standard labeled (physical characteristics are on the label), you need only specify INFILE in the first FILE statement. Record characteristics and file attributes are resolved by VISION:Results, eliminating the need to code them in either the first FILE statement or the DCB parameters in the JCL.

```
//OSUT0110 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=TAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL001
//OUTFILE DD DSN=TAPEOUT,DISP=(,KEEP),UNIT=TAPE,VOL=SER=DYL002
//SYSPRINT DD SYSOUT=A
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,11),RLSE)
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
FILE OUTFILE OUTPUT FROM INFILE FB 80 1600
/*
```

Figure 753 Standard Label Tape to Standard Label Tape

Reblocking

```
//OSUT0111 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=TAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL111
//OUTFILE DD DSN=TAPEOUT,DISP=(,KEEP),UNIT=TAPE,VOL=SER=DYL112,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600)
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE FB 80 1600
FILE OUTFILE OUTPUT FROM INFILE FB 80 3200
/*
```

Figure 754 Standard Label Tape to Standard Label Tape (Reblocking)

As in the previous example, neither the DCB parameters in the JCL nor the FILE statement input file description need to be specified; however, the physical attributes of the file have been included in the FILE statement. The order of precedence in overriding specifications is:

- FILE parameter
- JCL DCB parameter
- File label

Variable # of Input Files

```
//OSUT0112 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=TAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL301
// DD DSN=TAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL302
//OUTFILE DD DSN=TAPEOUT,DISP=(,KEEP),UNIT=TAPE,VOL=SER=DYL305
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
FILE OUTFILE OUTPUT FROM INFILE FB 80 1600
/*
```

Figure 755 Standard Label Tape to Standard Label Tape (Variable # of Input Files)

Variable-Spanned Records with Selection

```
//OSUT0114 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=TAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL501,
//OUTFILE DD DSN=TAPEOUT.VARSPAN,DISP=(NEW,KEEP),UNIT=TAPE,
// VOL=SER=DYL502
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
```

Figure 756 Standard Label Tape to Standard Label Tape (Variable-Spanned Records with Selection) (Page 1 of 2)

```
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE SB 160 324 LENGTH INRCSZ
FIRSTBYTE 1 NU
FILE OUTFILE OUTPUT FROM INFILE SB 160 324
IF FIRSTBYTE EQ 3 REJECT ENDIF
/*
```

Figure 756 Standard Label Tape to Standard Label Tape (Variable-Spanned Records with Selection) (Page 2 of 2)

You can assign a data name for LENGTH for the variable-spanned input file; however, you do not have to define the field. You set criteria to determine selection of particular records. This example rejects records with a 3 in the first byte and writes all others to the output file.

Unlabeled Tape to Standard Label Tape

```
//OSUT0114 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=NLTAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL155,
// LABEL=(,NL),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//OUTFILE DD DSN=TAPEOUT,DISP=(,KEEP),UNIT=TAPE,VOL=SER=DYL160
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
FILE OUTFILE OUTPUT FROM INFILE FB 80 800
/*
```

Figure 757 Unlabeled Tape to Standard Label Tape

All the attributes were specified in INFILE DD statements.

Standard Label Tape to Disk

Sequential Data Set

```
//OSUT0120 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=TAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL200
//OUTFILE DD DSN=DYL.DISKOUT,SPACE=(TRK,(5,5),RLSE),UNIT=3390,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600),DISP=(,CATLG,DELETE)
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
FILE OUTFILE OUTPUT FROM INFILE
/*
```

Figure 758 Standard Label Tape to Disk (Sequential Data Set)

ISAM, Blocked

```
//OSUT0121 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=TAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL200
//OUTFILE DD DSN=ISMA.BKLD (PRIME),DISP=(,CATLG,DELETE),
// UNIT=3350,VOL=SER=CUST01,SPACE=(CYL,1),
// DCB=DSORG=IS
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
FILE OUTFILE OUTPUT FROM INFILE FB 120 1200
ISAM KEYLEN 3 KEYLOC 1
/*
```

Figure 759 Standard Label Tape to Disk (ISAM, Blocked)

ISAM, Unblocked, RKP=0

```
//OSUT0122 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=TAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL200
//OUTFILE DD DSN=ISAMOUT.UNBLKD (PRIME),DISP=(,CATLG,DELETE),
// UNIT=3350,VOL=SER=CUST01,SPACE=(CYL,1),DCB=DSORG=IS
//WORKFL DD DUMMY
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
INRECORD 80 INKEY 3 1

FILE WORKFL FB 83 83
WRKKEY 3 1 WRKRCD 80 R

FILE OUTFILE OUTPUT FROM WORKFL F 80
ISAM KEYLEN 3 KEYLOC 1

MOVE INKEY TO WRKKEY
MOVE INRECORD TO WRKRCD
/*
```

Figure 760 Standard Label Tape to Disk (ISAM, Unblocked, RKP=0)

The key must precede the record when creating an unblocked file with RKP=0. The key is moved from the INFILE file to the WORKFL file. The data record INRECORD is moved to the WRKRCD record (field length of 80). The OUTFILE is written from the WORKFL file.

ISAM, Unblocked, RKP Not Equal to 0

```
//OSUT0123 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=TAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL200
//OUTFILE DD DSN=ISAMOUT.BKLD(PRIME),DISP=(,CATLG,DELETE),
// UNIT=3350,VOL=SER=CUST01,SPACE=(CYL,1),DCB=DSORG=IS
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
FILE OUTFILE OUTPUT FROM INFILE F 80 80
ISAM KEYLEN 1 KEYLOC 4
/*
```

Figure 761 Standard Label Tape to Disk (ISAM, Unblocked, RKP Not Equal to 0)

Standard Label Tape to Card Images

```
//OSUT0130 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=TAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL301
//OUTFILE DD SYSOUT=B
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
FILE OUTFILE OUTPUT FROM INFILE
F 80 80
/*
```

Figure 762 Standard Label Tape to Card Images

When creating punched output, remember to assign the proper punch SYSOUT class on the OUTFILE DD statement. By inserting PRINT INFILE in the OPTION statement, you can cause the contents of the INFILE to be printed as the file is being written to cards. z/OS unblocks OUTFILE when going to SYSOUT=B.

Standard Label Tape to Printer

Fixed-Length Records, Print Hex and Graphics

```
//OSUT0140 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=TAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL250
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
```

Figure 763 Standard Label Tape to Printer (Fixed-Length Records, Print Hex and Graphics) (Page 1 of 2)

```

*****
FILE INFILE
HEXPRINT INFILE
*
* or
*
OPTION HEXPRINT INFILE
FILE INFILE
/*

```

Figure 763 Standard Label Tape to Printer (Fixed-Length Records, Print Hex and Graphics) (Page 2 of 2)

Variable-Length Records, Print Hex and Graphics

```

//OSUT0141 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=VLTAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL275
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
*
* To Print Variable Records - Data Portion Only
*
FILE INFILE VB 160 LENGTH INLEN
RECFIELD 160
HEXPRINT RECFIELD LENGTH INLEN
*
* To Print Variable Records Including RDW Prefix
*
FILE INFILE
HEXPRINT INFILE
*
* or
*
OPTION HEXPRINT INFILE
FILE INFILE
/*

```

Figure 764 Standard Label Tape to Printer (Variable-Length Records, Print Hex and Graphics)

Variable-Spanned Records, Print Graphics Only

```
//OSUT0142 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=VARSPAN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL555
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
*
* To Print Variable Records - Data Portion Only
*
FILE INFILE SB 800 LENGTH INREC
  RECFIELD 800
PRINT RECFIELD LENGTH INREC
*
* To Print Variable Records Including RDW Prefix
*
FILE INFILE
PRINT INFILE
*
* or
*
OPTION PRINT INFILE
FILE INFILE
/*
```

Figure 765 Standard Label Tape to Printer (Variable-Spanned Records, Print Graphics Only)

Undefined Length Records, Print Hex and Graphics

```
//OSUT0143 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=VARSPAN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL555
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE U 2000
HEXPRINT INFILE
*
* or
*
OPTION HEXPRINT INFILE
FILE INFILE U 2000
/*
```

Figure 766 Standard Label Tape to Printer (Undefined Length Records, Print Hex and Graphics)

The characters per block of the FILE statement should contain a value that is at least as large as the largest block on the tape.

Partial Records, Print Graphics Only

```
//OSUT0144 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=TAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL144
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
*
* Print First 40 Bytes of Each Record
*
FILE INFILE
PRINT INFILE 40
*
* Print a Field Within a Record
*
FILE INFILE
INRECORD 80 INCREC1 40 1
PRINT INCREC1
*
* Print Partial or Multiple Fields Within a Record
*
FILE INFILE
INRECORD 80 INCREC1 40 1
PRINT INCREC1 LENGTH 25
/*
```

Figure 767 Standard Label Tape to Printer (Partial Records, Print Graphics Only)

Selected Records

```
//OSUT0145 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=TAPEIN,DISP=OLD,UNIT=TAPE,VOL=SER=DYL145
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
NUMBR 3 9 NU
IF NUMBR EQ 222 PRINT INFILE
ELSE REJECT ENDIF
/*
```

Figure 768 Standard Label Tape to Printer (Selected Records)

In this example, only those records with 222 in bytes 9-11, which are given the data name NUMBR, are to be printed. Selection is based upon the criteria set by you.

Report Tape with ASA Carriage Control

```
//OSUT0146 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=REPTAP,DISP=OLD,UNIT=TAPE,VOL=SER=DYL146
//OUTFILE DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330)
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
FILE OUTFILE OUTPUT FROM INFILE
/*
```

Figure 769 Standard Label Tape to Printer (Report Tape with ASA Carriage Control)

It is important that the record format attribute (RECFM=FBA) be put in the DCB parameter of the OUTFILE DD statement; otherwise, the carriage control bit is not set in the DCB. If the input report tape has machine carriage control, code FBM rather than FBA in the RECFM parameter of the DCB.

Disk to Standard Label Tape

```
//OSUT0210 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=DYDISKIN,DISP=OLD,UNIT=3390,VOL=SER=210210
//OUTFILE DD DSN=TAPEOUT,DISP=(,KEEP),UNIT=TAPE,VOL=SER=DYL210
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
FILE OUTFILE OUTPUT FROM INFILE FB 80 1600
/*
```

Figure 770 Disk to Standard Label Tape

If the input data set (INFILE) is not a cataloged disk data set, the UNIT and VOL=SER parameters (which vary with each installation) must be coded.

Disk (ISAM) to Standard Label Tape

```
//OSUT0211 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=ISAM.DATA,DISP=OLD,UNIT=3350,VOL=SER=TSO888,
//DCB=DSORG=IS
//OUTFILE DD DSN=ISAMTAPE,DISP=(,KEEP),UNIT=TAPE,VOL=SER=DYL211,
//DCB=(REF=CFM=FB,LRECL=120,BLKSIZE=1200)
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE ISAM
FILE OUTFILE OUTPUT FROM INFILE
/*
```

Figure 771 Disk (ISAM) to Standard Label Tape

Disk (ISAM) to Disk (ISAM)

Reorganize ISAM File

```
//OSUT0220 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1    EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE   DD DSN=ISAM.DATA,DISP=OLD,UNIT=3350,VOL=SER=TSO888,
//          DCB=DSORG=IS
//OUTFILE  DD DSN=ISAM.REORG(PRIME),VOL=SER=TSO999,UNIT=3350,
//          SPACE=(CYL,1),DCB=DSORG=IS,DISP=(,CATLG,DELETE)
//SYS004   DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE ISAM
FILE OUTFILE OUTPUT FROM INFILE FB 120 1200 ISAM
      KEYLEN 3  KEYLOC 1
/*
```

Figure 772 Disk (ISAM) to Disk (ISAM) (Reorganize ISAM File)

Disk (Sequential) to Disk

Partitioned Data Set—Adding or Replacing a Member in an Existing Library

```
//OSUT0221 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1    EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE   DD DSN=DYL.ABCMEM,DISP=OLD
//OUTFILE  DD DSN=DYL.SOURCELIB(ABCMEM),DISP=OLD
//SYS004   DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
FILE OUTFILE OUTPUT FROM INFILE
/*
```

Figure 773 Disk (Sequential) to Disk (Partitioned Data Set—Adding or Replacing a Member in an Existing Library)

Use DISP=OLD in the OUTFILE DD statement when replacing or adding a member to a PDS so that it cannot be accessed by another job while it is being modified.

Disk to Disk

Undefined Copy

```
//OSUT0223 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=DISKIN.UNDEF,DISP=SHR
//OUTFILE DD DSN=DISKOUT.UNDEF,DISP=(,CATLG,DELETE),
// UNIT=3390,SPACE=(TRK,5,RLSE),VOL=SER=TSO444
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE U 400
FILE OUTFILE OUTPUT FROM INFILE U 400
/*
```

Figure 774 Disk to Disk (Undefined Copy)

The block size should be as large as or larger than the largest block on the file.

Copy Variable-Spanned Records

```
//OSUT0224 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=DISKIN.VARSP,DISP=SHR
//OUTFILE DD DSN=DISKOUT.VARSP,DISP=(,CATLG,DELETE),
// UNIT=3390,SPACE=(TRK,5,RLSE),VOL=SER=TSO444
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
FILE OUTFILE OUTPUT FROM INFILE SB 84 424
/*
```

Figure 775 Disk to Disk (Copy Variable-Spanned Records)

Disk to Card Images

```
//OSUT0230 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=DYL.DISKIN.DATA,DISP=SHR
//OUTFILE DD SYSOUT=B
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
FILE OUTFILE OUTPUT FROM INFILE FB 80 800
/*
```

Figure 776 Disk to Card Images

z/OS unblocks OUTFILE when output goes to SYSOUT=B.

Disk to Printer

Print Graphic Only

```
//OSUT0240 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=DYL.DISKIN.DATA,DISP=SHR
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
PRINT INFILE
*
* or
*
OPTION PRINT INFILE
FILE INFILE
/*
```

Figure 777 Disk to Printer (Print Graphic Only)

Limited Output, Print Hex and Graphics

```
//OSUT0241 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=DYL.DISKIN.DATA,DISP=SHR
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE COUNT INFCNT
IF INFCNT GT 10 STOP ENDIF
HEXPRINT INFILE
*
* or
*
OPTION HEXPRINT INFILE
FILE INFILE COUNT INFCNT
IF INFCNT GT 10 STOP ENDIF
/*
```

Figure 778 Disk to Printer (Limited Output, Print Hex and Graphics)

In this example, the first 10 records are printed. The data name, INFCNT, assigned to COUNT keeps track of the number of records processed.

ISAM, Sequential Read

```
//OSUT0242 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE DD DSN=DYL.DISKIN.ISAM,DISP=SHR,DCB=DSORG=IS
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE ISAM
PRINT INFILE
*
* or
*
OPTION PRINT INFILE
FILE INFILE ISAM
/*
```

Figure 779 Disk to Printer (ISAM, Sequential Read)

ISAM, Random Read

```
//OSUT0243 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//ISAMFL DD DSN=DYL.DISKIN.ISAM,DISP=SHR,DCB=DSORG=IS
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
OPTION PRINT ISAMFL
FILE SYSIN CARDS IKEY 3 1
FILE ISAMFL FB 70 ISAM RANDOM SKEY
KEYLEN 3 KEYLOC 1 STATUS ISTAT
READ SYSIN
IF SYSIN EQ X'FF' GOTO ENDJOB ENDIF
MOVE IKEY TO SKEY
READ ISAMFL
IF ISTAT EQ 'Y' ACCEPT ENDIF
MOVE 100 TO DYLRETURN
ENDJOB: STOP
FIN
Insert Instream Data of Record Keys
/*
```

Figure 780 Disk to Printer (ISAM, Random Read)

Specific [ISAM](#) records are to be retrieved randomly and printed. The instream data file is placed after the VISION:Results FIN statement in the SYSIN data stream. The [ISAM](#) file to be read randomly is the ISAMFL file.

List Members of a Partitioned Data Set

```
//OSUT0244 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1   EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//INFILE   DD DSN=DYL.MACLIB(XYZ),DISP=SHR
//SYS004   DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
*****
* VISION:Results Program Statements *
*****
FILE INFILE
PRINT INFILE
*
* or
*
OPTION PRINT INFILE
FILE INFILE
/*
```

Figure 781 Disk to Printer (List Members of a Partitioned Data Set)

Instream to Standard Label Tape

```
//OSUT0310 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1   EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//OUTFILE  DD DSN=TAPEOUT,DISP=(,KEEP),UNIT=TAPE,
//          VOL=SER=DYL777,
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYS004   DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
*****
* VISION:Results Program Statements *
*****
FILE SYSIN CARDS
FILE OUTFILE OUTPUT FROM SYSIN
FIN
    Insert Instream Data File
/*
```

Figure 782 Instream to Standard Label Tape

Variable-Length Records

```
//OSUT0311 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1   EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//OUTFILE  DD DSN=VARLEN.TAPEOUT,DISP=(,KEEP),VOL=SER=DYL501,
//          DCB=(RECFM=VB,LRECL=84,BLKSIZE=844),UNIT=TAPE
//SYS004   DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
*****
* VISION:Results Program Statements *
*****
FILE SYSIN CARDS
    INRECORD 80 1
FILE OUTFILE OUTPUT FROM OUTFILE VB 80 LENGTH RECSIZE
    OURECORD 80 1
MOVE 80 TO RECSIZE
MOVE INRECORD TO OURECORD
FIN
    Insert Instream Data File
/*
```

Figure 783 Instream to Standard Label Tape (Variable-Length Records)

The data name, RECSIZE, assigned to LENGTH in the FILE statement keeps track of the length of the variable-length record.

Reformatting Instream Data

```
//OSUT0312 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//OUTFILE DD DSN=REFMT.TAPEOUT,DISP=(,KEEP),VOL=SER=DYL456,
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),UNIT=TAPE
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE SYSIN INPUT CARDS
      INNUMB 5 INNAME 20 INCITY 20 INADDR 25
FILE OUTFILE OUTPUT FROM OUTFILE FB 80
      OUNAME 20 OUNUMB 5 OUADDR 25 OUCITY 20

MOVE INNUMB TO OUNUMB
MOVE INNAME TO OUNAME
MOVE INADDR TO OUADDR
MOVE INCITY TO OUCITY
FIN
      Insert Instream Data File
/*
```

Figure 784 Instream to Standard Label Tape (Reformatting Instream Data)

Although the logical record length is defined in the DCB parameter of the OUTFILE DD statement, you have to define the record size in the FILE statement to create an output buffer.

Instream to Unlabeled Tape

Instream Data Contains JCL

```
//OSUT0313 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//OUTFILE DD DSN=NOLABEL,DISP=(,KEEP),UNIT=TAPE,VOL=SER=DYL987,
//          LABEL=(,NL)
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD DATA,DLM=EE
*****
* VISION:Results Program Statements *
*****
FILE SYSIN CARDS
FILE OUTFILE OUTPUT FROM SYSIN F 80 80
FIN
      Insert Instream Data File
EE
```

Figure 785 Instream to Unlabeled Tape (Instream Data Contains JCL)

Because the input contains delimiter statements (/* or //), some other delimiter must be used to signal the end of the instream data (in this case, EE).

Instream to Disk

Sequential

```
//OSUT0320 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1   EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//OUTFILE DD DSN=DYL.DISKOUT,DISP=(,CATLG,DELETE),UNIT=3390,
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600),
//          SPACE=(TRK,5,RLSE)
//SYS004  DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN   DD *
*****
* VISION:Results Program Statements *
*****
FILE SYSIN CARDS
FILE OUTFILE OUTPUT FROM SYSIN FB 80 1600
FIN
    Insert Instream Data File
/*
```

Figure 786 Instream to Disk (Sequential)

ISAM, Blocked

```
//OSUT0321 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1   EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//OUTFILE DD DSN=DYL.BKLD.ISAM(PRIME),DISP=(,CATLG,DELETE),
//          DCB=DSORG=IS,SPACE=(CYL,1),UNIT=3350,VOL=SER=TSO900
//SYS004  DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN   DD *
*****
* VISION:Results Program Statements *
*****
FILE SYSIN CARDS
FILE OUTFILE OUTPUT FROM SYSIN FB 70 700 ISAM
    KEYLEN 3 KEYLOC 2
FIN
    Insert Instream Data File
/*
```

Figure 787 Instream to Disk (ISAM, Blocked)

Unblocked ISAM, RKP=0

```
//OSUT0322 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//OUTFILE DD DSN=DYL.UNBLKD.ISAM(PRIME),DISP=(,CATLG DELETE),
//        SPACE=(CYL,1),UNIT=3350,VOL=SER=TSO456
//WORKFL DD DUMMY
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE SYSIN CARDS
  INKEY 3 INRECORD 80 1
FILE WORKFL FB 83 83
  WKKEY 3 WKRECORD 80 4
FILE OUTFILE OUTPUT FROM WORKFL F 80 ISAM
  KEYLEN 3 KEYLOC 1
MOVE INKEY TO WKKEY
MOVE INRECORD TO WKRECORD
FIN
  Insert Instream Data File
/*
```

Figure 788 Instream to Disk (Unblocked ISAM, RKP=0)

When creating an unblocked [ISAM](#) file whose key starts in the first byte of the record (RKP=0), you must ensure that the key precedes the data portion of the record even though the key can also be contained in the data portion. You have to create a work file that has the same record length plus the size of the key.

Unblocked ISAM with RKP Not Equal to 0

```
//OSUT0323 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//OUTFILE DD DSN=DYL.UNBLKD.ISAM(PRIME),DISP=(,CATLG,DELETE),
//        DCB=(RECFM=F,LRECL=80,BLKSIZE=80,RKP=1,KEYLEN=2,DSORG=IS),
//        VOL=SER=TSO500,UNIT=3350,SPACE=(CYL,1)
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE SYSIN CARDS
FILE OUTFILE OUTPUT FROM SYSIN ISAM
FIN
  Insert Instream Data File
/*
```

Figure 789 Instream to Disk (Unblocked ISAM with RKP Not Equal to 0)

Instream to Card Images

```
//OSUT0330 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//OUTFILE DD SYSOUT=B
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
```

Figure 790 Instream to Card Images

```

*****
* VISION:Results Program Statements *
*****
OPTION PRINT SYSIN
FILE SYSIN CARDS
FILE OUTFILE OUTPUT FROM SYSIN FB 80 1680
FIN
  Insert Instream Data File
/*

```

Figure 790 Instream to Card Images

z/OS unblocks OUTFILE when going to SYSOUT=B. In addition to creating a duplicate card deck, this program prints the instream data file in graphics format.

Instream to Printer

```

//OSUT0340 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE SYSIN CARDS
PRINT SYSIN
*
* or
*
OPTION PRINT SYSIN
FILE SYSIN CARDS
FIN
  Insert Instream Data File
/*

```

Figure 791 Instream to Printer

VSAM—Define KSDS Fixed Format File Using IDCAMS

```
//OSUT0410 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=IDCAMS,REGION=600K,TIME=(,10)
//SYSPRINT DD SYSOUT=A
//VOL4 DD UNIT=3390,VOL=SER=VSAM01,DISP=OLD
//SYSIN DD *
*****
* IDCAM Statements *
*****
DELETE DYLVS.FKSDS1.ALL -
        CLUSTER -
        FILE (FKSDS1) -
        PURGE -
        CATALOG (UCAT.VVSAM01)
DEFINE CLUSTER (NAME (DYLVS.FKSDS1.ALL) -
        FILE (FKSDS1) -
        VOL (VSAM01) -
        TRACKS (4,1) -
        KEYS (5,2) -
        RECSZ (40,40) -
        SPEED) -
        DATA (NAME (DYLVS.FKSDS1.DATA) -
        CISZ (2048) -
        FREESPACE (50,25)) -
        INDEX (NAME (DYLVS.FKSDS1.INDEX) -
        CISZ (5121) -
        CATALOG (UCAT.VVSAM01))
        IF LASTCC LT 12 THEN -
        LISTCAT ENTRIES (DYLVS.FKSDS1.ALL) ALL

/*
```

Figure 792 VSAM—Define KSDS Fixed Format File Using IDCAMS

A VSAM KSDS fixed format file is being defined with a record length of 40. The key is 5 bytes long, beginning in the third byte of the record.

VSAM KSDS—Tape to Disk

Create/Restore a Fixed KSDS VSAM File from a Fixed Format Sequential File

```
//OSUT0411 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//SEQIN DD DSN=DYL.SEQVS,DISP=OLD,
//        UNIT=TAPE,VOL=SER=DYL189
//FKSDS DD DSN=DYLVS.FKSDS1.ALL,DISP=OLD
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE SEQIN
FILE FKSDS KSDS OUTPUT FROM SEQIN F

/*
```

Figure 793 VSAM KSDS—Tape to Disk (Create/Restore a Fixed KSDS VSAM File from a Fixed Format Sequential File)

If you want to write over an existing VSAM file, the REUSE option must have been specified when the file was defined in an IDCAMS program. The keyword REUSE must also be supplied in the output file statement of the VISION:Results program. If these two requirements are not met, the file being restored must be deleted and redefined in an IDCAMS run before recreating.

VSAM KSDS—Disk to Tape

Backup a Fixed KSDS VSAM File to Another File

```
//OSUT0412 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//FKSDSI DD DSN=DYLV.S.FKSDS1.ALL,DISP=OLD
//SEQOUT DD DSN=DYL.SEQVS,DISP=(,KEEP),
// UNIT=TAPE,VOL=SER=DYL911
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE FKSDSI KSDS F
FILE SEQOUT OUTPUT FROM FKSDSI FB 40 4000
/*
```

Figure 794 VSAM KSDS—Disk to Tape (Backup a Fixed KSDS VSAM File to Another File)

VSAM KSDS—Disk to Printer

Fixed KSDS VSAM File, Print Hex and Graphics

```
//OSUT0413 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//FKSDS DD DSN=DYLV.S.FKSDS1.ALL,DISP=OLD
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
OPTION HEXPRINT
FILE FKSDS KSDS F
/*
```

Figure 795 VSAM KSDS—Disk to Printer (Fixed KSDS VSAM File, Print Hex and Graphics)

Retrieve, Sort in Another Sequence, and Print

```
//OSUT0414 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//FKSDS DD DSN=DYLV.S.FKSDS1.ALL,DISP=OLD
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
//SORTWK01 DD UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SYSOUT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
```

Figure 796 VSAM KSDS—Disk to Printer (Retrieve, Sort in Another Sequence, and Print)

```
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE FKSDS KSDS F 40
  SORTFIELD 3 21
SORT FKSDS USING SORTFIELD
PRINT FKSDS
/*
```

Figure 796 VSAM KSDS—Disk to Printer (Retrieve, Sort in Another Sequence, and Print)

VSAM—Define ESDS Fixed Format File Using IDCAMS

```
//OSUT0420 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=IDCAMS,REGION=600K,TIME=(,10)
//SYSPRINT DD SYSOUT=A
//VOL4 DD UNIT=3390,VOL=SER=VSAM01,DISP=OLD
//SYSIN DD *
*****
* IDCAM Statements *
*****
DELETE DYLVS.FESDS1 ALL -
  CLUSTER -
  FILE (FESDS1) -
  PURGE -
  CATALOG (UCAT.VVSAM01)
DEFINE CLUSTER (NAME (DYLVS.FESDS1.ALL) -
  FILE (FESDS1) -
  VOL (VSAM11) -
  TRACKS (2,1) -
  NONINDEXED -
  RECSZ (29,29) -
  SPEED) -
  DATA (NAME (DYLVS.FESDS1.DATA) -
  CATALOG (UCAT.VVSAM01))
IF LASTCC LT 12 THEN -
  LISTCAT ENTRIES (DYLVS.FESDS1.ALL) ALL
/*
```

Figure 797 VSAM—Define ESDS Fixed Format File Using IDCAMS

A VSAM ESDS fixed format file is being defined with a record length of 29.

VSAM ESDS—Tape to Disk

Create/Restore a Fixed ESDS VSAM File from a Fixed Format Sequential File

```
//OSUT0421 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//SEQIN DD DSN=DYL.SEQVS,DISP=OLD,UNIT=TAPE,
// VOL=SER=DYL745
//FESDS DD DSN=DYLVS.FESDS1.ALL,DISP=OLD
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
```

Figure 798 VSAM ESDS—Tape to Disk (Create/Restore a Fixed ESDS VSAM File from a Fixed Format Sequential File) (Page 1 of 2)

```

* VISION:Results Program Statements *
*****
FILE SEQIN
FILE FESDS ESDS OUTPUT FROM SEQIN F
/*

```

Figure 798 VSAM ESDS—Tape to Disk (Create/Restore a Fixed ESDS VSAM File from a Fixed Format Sequential File) (Page 2 of 2)

If you want to write over an existing VSAM file, the REUSE option must have been specified when the file was defined in an IDCAMS program. The keyword REUSE must also be supplied in the output file statement of the VISION:Results program. If these two requirements are not met, the file being restored must be deleted and redefined in an IDCAMS run before recreating. Otherwise, the records are added to the end of the existing ESDS file.

VSAM ESDS—Disk to Tape

Backup a Fixed ESDS VSAM File to Another File

```

//OSUT0422 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//FESDSI DD DSN=DYLV.S.FESDS1.ALL,DISP=OLD
//SEQOUT DD DSN=DYL.SEQVS,DISP=(,KEEP),
// UNIT=TAPE,VOL=SER=DYL654
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE FESDS ESDS F
FILE SEQOUT OUTPUT FROM FESDS FB 29 2900
/*

```

Figure 799 VSAM ESDS—Disk to Tape (Backup a Fixed ESDS VSAM File to Another File)

VSAM ESDS—Disk to Printer

Fixed ESDS VSAM File, Print Hex and Graphics

```

//OSUT0423 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//FESDS DD DSN=DYLV.S.FESDS1.ALL,DISP=OLD
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****

```

Figure 800 VSAM ESDS—Disk to Printer (Fixed ESDS VSAM File, Print Hex and Graphics)

```
* VISION:Results Program Statements *
*****
FILE FESDS ESDS F
  RELBYTE RBA

PROCEDURE:
  MOVE RBA TO DYLPRTCOMM
  PRINT FESDS
/*
```

Figure 800 VSAM ESDS—Disk to Printer (Fixed ESDS VSAM File, Print Hex and Graphics)

The entire file is read and printed in hex and graphics with the RBA printing to the left of the record.

Retrieve, Sort in Another Sequence, and Print

```
//OSUT0424 JOB (SWDEV,123456,DYL,55),'RJE CA UTL'
//STEP1 EXEC PGM=DYL280,REGION=600K,TIME=(,10)
//FESDS DD DSN=DYLV.S.FESDS1.ALL,DISP=OLD
//SYS004 DD UNIT=SYSDA,SPACE=(TRK,(10,1),RLSE)
//SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
//SORTWK01 DD UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(TRK,(5),,CONTIG)
//SYSOUT DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
*****
* VISION:Results Program Statements *
*****
FILE FESDS ESDS F 29
  SORTFIELD 11 2

SORT FESDS USING SORTFIELD
PRINT FESDS
/*
```

Figure 801 VSAM ESDS—Disk to Printer (Retrieve, Sort in Another Sequence, and Print)

Index

Symbols

\$DEFAULT command • 328
\$DEND command • 328
\$ELSE command • 330
\$IFE command • 329
* multiply arithmetic operator • 231
** exponentiation arithmetic operator • 231
+ add arithmetic operator • 231
- subtract arithmetic operator • 231
/ delimiter statement • 816
/ divide arithmetic operator • 231
/* delimiter statement • 816
// delimiter statement • 816
@ followed by a data name • 738
@ sign • 738, 744
@(ASIS) • 736
@(ASIS) line • 744
@(ASIS) quotation marks • 745
@(CSnn) • 736
@(Eject) • 736
@(NEWPARA) • 737
@(SPn) line spacing keyword • 735
@dataname variable information • 734
@LTDnn detail text statement • 734
@nnn • 737
@nnn positioning • 745

A

A edit code • 314, 739
ABEND 0C4 (unpredictable results) • 301
ABSOLUTE field • 228

ACCEPT • 155, 159, 174, 177
access methods • 567
 random access • 567
 sequential access • 567
access mode coded • 801
adding a field to a report • 213
adding a member in an existing library • 811
aligning data names and literals
 at detail time • 402
 at ON CHANGE IN • 403
 at ON FINAL • 403
ALLOCATE command • 124, 317, 323
allocation, dynamic • 124, 317
AMODE(31) • 293
analysis, trend line • 767, 770
API completion and reason codes • 449
area specified • 802
areas, problem • 269
ARFILE input file statement • 748
arithmetic operations
 exponentiation • 233
 indexing • 232
 order of computation • 233
 processing priorities • 233
 restrictions • 235
 rounding • 232
arithmetic operators • 231
 * multiply • 231
 ** exponentiation • 231
 + add • 231
 - subtract • 231
 / divide • 231
array
 allocating memory • 705
 Array Element Field Definition(s) • 722
 arrayname • 721

- DIM (DIMENSION) • 722
- error return codes • 724
- F element-length • 721
- FILL fillchar • 722
- handling • 705
- loading of • 722
- RETAIN|NORETAIN • 722
- retrieval from • 723
- STATUS • 721
- syntax • 721
- USE • 723
- user return codes • 724
- Array Element Field Definition(s) • 722
- arrayname • 721
- ASA carriage control • 727, 732, 765, 810
- ASIS (@ASIS) • 749
- assembler
 - example • 286
 - subroutine considerations • 285
- asterisks (*) used for comments • 57
- at detail time, aligning data names and literals • 402
- at ON CHANGE IN, aligning data names and literals • 403
- at ON FINAL, aligning data names and literals • 403
- AUDPRINT/SYS009 • 732, 765
- AUTOLINK link editor feature • 283, 284
- automatic cycle • 171
 - ON END OF INPUT • 174
 - structured mode • 173
 - with SORT • 175
 - without SORT • 173

B

- B edit code • 316, 741
- backup a fixed KSDS VSAM file to another file • 821, 823
- Backus-Naur-Form (BNF) notation • 103
- BDAM • 565, 597
 - example • 600, 601, 602, 604, 605, 606, 607, 608
 - file create • 573
 - I/O statistics • 567
 - numeric keys • 599, 600

- packed keys • 599, 600
- random read • 580
- random update • 590
- record access • 566
- record addressing schemes • 566
- sequential create • 573
- sequential read • 586
- skipping blocks • 579
- skipping records • 577
- syntax • 565
- undefined record length • 576
- variable-length records • 597

- BETWEEN in TWOUP keyword • 732, 759
- BI (binary data) • 34, 127, 232
- binary data (BI) • 127, 232
- BINSEARCH
 - binary search • 712
- BINSEARCH binary search • 712
- BITS, COMBINE • 224, 225, 226, 527, 561
- BLANK or BLANKS • 45
- block size • 186, 801
- blocked and unblocked files in ISAM • 558
 - random update • 560
 - sequential load or extend • 559
 - sequential or skip-sequential read • 558
- blocked data • 817
- blocked ISAM • 805
- BNF (Backus-Naur-Form) • 103
- break, control • 407
- BY statement • 772, 773
- BYyyy in DYLTRxx keyword • 732

C

- CA
 - contacting Technical Support • 30
- CA-Librarian • 337, 338
- CA-Panvalet • 337
- CALL • 46, 48
- CALL command • 275
 - amode(31) • 293
 - assembler subroutine considerations • 285
 - COBOL subroutine considerations • 281
 - example starting a PL/I external subroutine •

-
- 288
 - EXIT module writing • 291
 - external subroutine considerations • 278
 - MODIFY module writing • 290
 - nK (size) parameter (VSE only) • 275
 - parameters examples • 277
 - PL/I subroutine considerations • 287
 - subroutine name position • 275
 - carriage control ASA keyword • 727, 732, 765
 - CASE • 162, 165, 166
 - ENDCASE • 166
 - EXITCASE • 167
 - nesting • 166
 - restrictions • 167
 - structure • 156
 - CH (character data) • 34
 - CHANGE IN, aligning data names and literals at ON • 403
 - CHANGE IN, ON • 297, 298, 409
 - characters per input line, number of • 59
 - class tests • 147
 - MIXED (bits mixed) • 150
 - negative value • 148
 - numeric value • 148
 - ONES (bits on) • 149
 - positive value • 147
 - CLOSE filename • 301
 - COBOL • 29, 209, 341
 - examples • 342
 - record descriptions • 340
 - replacing literal • 343
 - replacing pseudo-text • 344
 - COBOL program • 284
 - OC1 problem • 284
 - OC4 problem • 284
 - COPY and COPYE examples • 340
 - example ANSI • 282
 - EXIT module, writing one • 291
 - MODIFY module, writing one • 290
 - problem areas • 284
 - subroutine called by VISION:Results • 283
 - subroutine considerations • 281
 - USING upon entering example • 281
 - z/OS example • 283
 - coding standards in structured programming • 159
 - column headings overriding defaults • 200
 - COLUMNS keyword in the OPTION statement example • 59
 - COMBINE BITS • 224, 225, 527, 561
 - example • 226
 - using AND • 225
 - using EXOR • 226
 - using OR • 224
 - combine fields using EXOR • 226
 - combined range and series compares • 147
 - combining compound compares • 152
 - commands
 - \$DEFAULT • 328
 - \$DEND • 328
 - \$ELSE • 330
 - \$IFE • 329
 - ACCEPT • 297
 - ALLOCATE • 124, 317, 323
 - CLOSE • 301
 - CONTROL • 194
 - COPY • 208, 209, 328, 340, 745, 797
 - COPY or COPYE for VSE • 337
 - COPY or COPYE for z/OS • 337
 - COPYC • 338, 745
 - COPYDB2 • 345
 - COPYE • 328, 340
 - COPYL • 338, 745
 - COPYN • 338
 - COPYP • 339, 745
 - DELETE • 716
 - FILE • 185
 - GOTO • 296
 - IF • 202
 - IF-OR • 204
 - LETTER • 729, 733, 738, 742, 749, 755
 - LIST • 188, 379
 - MOVE CORRESPONDING • 96
 - NEXT • 297
 - PERFORM • 161, 299
 - QUIT • 298
 - QUITALL • 299
 - READ • 301
 - REGRESSION • 773, 775
 - REJECT • 298
 - RETAIN • 131
 - REUSE • 127
 - SORT • 192, 797
 - STOP • 298
-

- STOPALL • 298
- TITLE • 200
- TRANSLATE • 309
- WRITE • 301
- commands for printing • 478
- comments including remarks • 57
- compares
 - combined range and series • 147
 - combining compound or multiple compares • 152
 - compound • 150
 - multiple • 150
 - range • 146
 - series • 147
- compile phase error messages • 323
- compound compares • 150
 - combining • 152
 - selecting and processing if all of several conditions are true • 152
 - selecting and processing if any of several conditions are true • 151
- confirmation notice, reserved words
 - DYLCNxxID field value • 731
 - DYLCNxxLNC • 756
 - DYLCNxxSW • 733
 - DYLEXTxx • 730
 - DYLEXTxx keyword • 731
- contacting CA
 - <http://ca.com/support> • 30
- contacting Technical Support • 30
- continuation rules example • 58
- CONTROL • 194, 195
- control break levels • 407
- CONTROL statement • 407
- CONVENTIONAL mode • 158
- conventions
 - naming data names • 58
 - naming field names • 58
- coordinates • 722
- COPY • 208, 797
- COPY command • 745
- COPY or COPYE command • 328, 340
 - COBOL example • 342
 - COBOL rules • 341
 - COPYDB2 • 345
 - COPYDB2 example • 349
 - COPYL • 338
 - example • 330, 331, 332, 334
 - for VSE • 337
 - for z/OS • 337
 - limitations • 336
 - rules • 336
 - z/OS JCL requirements • 337
- COPY statements to include file definitions • 800
- copy variable-spanned records • 812
- COPYC command • 338, 745
- COPYDB2 command • 345
 - example • 349
 - JCL DB2 overrides • 346
 - requirements • 346
 - retrieving fields of a DB2 table • 345
- COPYDB2 requirements • 346
- COPYL command • 338, 745
- COPYN command • 338
- COPYP command • 339, 745
- COPYx command general considerations • 336
- CORRESPONDING, MOVE command • 96
- COUNT keyword • 567
- count of records • 567
- create
 - fixed ESDS VSAM • 822
 - fixed KSDS VSAM • 820
 - sequential BDAM file • 573
- cuu defined • 273
- CYLOFL keyword, optional for VSE • 538

D

- D edit code • 316
- D4 edit code • 316
- DA edit code • 315
- data
 - interval selection of • 154
 - random selection of • 154
- data and schema documents generation • 439
- data names • 32
 - and literals at detail time, aligning • 402
 - and literals at ON CHANGE IN, aligning • 403
 - and literals at ON FINAL, aligning • 403

- characters allowed • 32
 - for LENGTH • 804
 - indexed • 380
 - number of decimals • 35
 - prefixing • 344
 - qualification of • 41, 93
 - sorting of • 257
 - VALUE • 35
- data types • 34
 - binary (BI) • 127, 232
 - numeric (NU) • 127, 232
 - packed (PD) • 127, 232
- DATAVER diskette option • 124
- DCB parameter • 801, 816
- DD statement • 743
- ddname the same as file name • 800
- ddnames • 732, 743
- DE edit code • 314
- decimal data position, PD (packed data) • 127, 232
- define ESDS fixed format file using IDCAMS • 822
- definition
 - of file that has program statements • 800
 - of files being processed • 800
 - of source PDS • 800
- DELETE command • 716
- delimiter statements • 816
- descriptor word, record • 481
- detail time, aligning data names and literals at • 402
- diagram, scatter • 767
- DIMENSION • 722
- DIMENSION|DIM • 722
- directing program execution • 295
 - ACCEPT • 297
 - ALLOCATE • 317
 - CLOSE filename • 301
 - commands • 296
 - ENDONE • 302
 - example • 296
 - GOTO • 296
 - indices • 306
 - NEXT • 297
 - ON END OF INPUT • 303
 - ON ONE • 302
 - PERFORM • 299
 - QUIT • 298
 - QUITALL • 299
 - READ filename • 301
 - REJECT • 298
 - STOP • 298
 - STOPALL • 298
 - TRANSLATE • 309
 - WRITE filename • 301
- disk (ISAM file) to disk (ISAM file) (reorganize ISAM file) • 811
- disk (ISAM file) to standard label tape • 810
- disk (sequential) to disk (partitioned) • 811
- DISK nnnn supported values • 123
- disk to card images • 812
- disk to disk
 - copy variable-spanned records • 812
 - undefined copy • 812
- disk to printer
 - (ISAM, random read) • 814
 - (ISAM, sequential read) • 814
 - (limited output, print hex and graphics) • 813
 - (list members of a partitioned data set) • 815
 - (print graphics only) • 813
- disk to standard label tape • 810
- DISK type option • 123, 538, 569, 571
- DISK xxxx in DYLTRxx keyword • 732
- DISKETTE
 - DATAVER (data verified) • 124
 - NOFEED (not placed in collector's bin) • 124
 - NUMFILES dynamically specified • 122
 - options of 3540 • 124
 - PROTECT (cannot be overwritten) • 124
 - SECURE (data is secured) • 124
 - VERIFY (perform operations specified) • 124
 - VOLVER (volume verified) • 124
- DLBL name keyword • 732
- DMAP option (data name map) • 97, 98
- DNA edit code • 315
- DNE edit code • 314
- DOUNTIL • 157, 160, 161, 168, 177
 - ENDDO • 169
 - EXITDO • 170
 - ITERATE • 171
 - nesting • 169

- restrictions • 170
- DOWHILE • 157, 160, 161, 168, 177
 - ENDDO • 169
 - EXITDO • 170
 - ITERATE • 171
 - nesting • 169
 - restrictions • 170
- DY282MAT (match statement) • 637
 - example • 644, 645, 646
 - match subroutine • 665
 - replacement routine • 674
 - USING DY282MAT • 642
- DY282MER (merge statement) • 637
 - parameters passed to • 682
 - replacement routine • 684
 - rule for • 659
 - USING DY282MER • 660
- DYCHANEL copy member • 745
- DYL280Rx phase name • 428
- DYLCENTRY1 • 45
- DYLCENTRY2 • 46
- DYLCNAME00 • 46
- DYLCNAME0A-0Z • 46
- DYLCNAME1-99 • 46
- DYLCNxxID keyword • 731
- DYLCNxxLNC special reserved word • 756
- DYLCNxxSW 1-byte field • 733
- DYLCOMMON • 46
- DYLCOMRG • 47
- DYLCOUNT00 • 47
- DYLCOUNT0A-0Z • 47
- DYLCOUNT1-99 • 47
- DYLDATE • 47
- DYLDATE reserved word • 748
- DYLDATE4 • 47
- DYLDATEPAG • 47
- DYLDATEPG4 • 48
- DYDLA • 48
 - DYLTDLI (IMS/DL/I database interface option) • 48
- DYLETIME • 48
- DYLEXTxx keyword • 730, 731
- DYLGREG • 48
- DYLGREG4 • 48
- DYLJULIAN • 49
- DYLJULIAN4 • 49
- DYLLINE • 49
- DYLNOTOT • 49
- DYLNRP • 50
- DYLONE • 50
- DYLPAGE • 50
- DYLPAGE_n • 50
- DYLPARM • 50
- DYLPARMLen • 51
- DYLPPAGE • 51
- DYLPRC_x subroutine module • 428
- DYLPRTCOMM • 53
- DYLPRTCOMM 10-byte comment field • 482
- DYLPRTMAXL • 53
- DYLPRTNUMB • 54
- DYLP RTPAGE • 54
- DYLREPMAXL • 54
- DYLRESET • 54
- DYLRETURN • 55
- DYLSYSDATE • 55
- DYLSYSPARM • 55
- DYLTIME • 55
- DYLTRAN • 56
- DYLTRAN modify translate table • 311
- DYLTRxx specified in WIDE keyword • 730, 732
- DYLUPSI • 56
- DYNAM (dynamically allocated file name) • 124, 317
- dynamic allocation • 124, 317
 - CALL • 323
 - diagnostic output • 323
 - error messages • 323
 - example • 321
 - HTML PDS • 367
 - HTML with HFS path • 366
 - JCL considerations • 318
 - LTH • 323
 - PICNSAVE • 322
 - REPORT • 322
 - rules for • 317

E

E edit code • 313

edit codes • 313, 739, 742

A • 314, 739

B • 316, 741

D • 316, 741

D4 • 316

DA • 315

DE • 314

DNA • 315

DNE • 314

E • 313, 739

F • 314, 739

G • 314, 739

H • 315, 740

J • 315, 740

K • 314, 739

L • 315, 740

NA • 314

NB • 316

NDA • 315

NDE • 314

NE • 313

NP • 316

NZ • 315

override defaults • 313, 739, 742

P • 316, 741

Q • 316, 741

S • 316, 742

U • 315, 740

V • 316, 742

W • 316, 741

X • 314, 740

Y • 313, 739

Z • 315, 740

EDIT command • 311, 312

listed fields edited automatically • 311

non-standard edits • 312

standard edits • 313

EE end of the instream data • 816

EJECT (@EJECT) parameter • 762

eject, page • 742, 756

ELSE • 160, 161, 163, 164, 165

END OF INPUT, ON • 303, 304, 305

END OF SORTING, ON • 244

ENDCASE • 166

ENDDO • 169

ENDIF • 160, 161, 163, 164

ENDLTD • 737

ENDLTD keyword • 738, 744, 749

ENDLTH keyword • 729

ENDONE (terminate statement) • 302

ENTRY keyword • 707

ENTRY table keyword • 707

equation, straight line • 768

ERASE keyword • 508

error analysis for source • 799

error handling, XML • 447

error messages

628E unable to allocate work area • 136

in compile phase • 323

error of the estimate, standard • 769

ESDS VSAM file backup to another file • 823

example FILE statement • 801

EXCLUSIVE OR (EXOR) command • 226

execution, directing program • 295

EXIT module • 134, 301

PARM • 292

STATUS • 292

writing • 291

EXIT modulename • 132, 291

restrictions • 133

exit point • 427

considerations • 428

exit subroutine letter writing • 727

EXIT-processed • 133

EXITCASE • 167

EXITDO • 170

EXOR (EXCLUSIVE OR) • 226

exponentiation (**) • 233

express an EXOR command • 226

extended format • 38

extended format of field definition • 38

Extensible Markup Language (See XML) • 431

EXTENTS extent type code • 538

F

F edit code • 314, 739

F element-length • 721

F entrylen keyword • 706

FCB DD parameter • 743

field

- adding to a report • 213

- definitions • 31

- length • 33

- moving to a subtotal • 222

- redefining • 34

field definition statements • 190

field definitions • 31, 187

- character values • 35

- column heading • 40

- data name • 32

- data type • 34

- edit code • 38

- extended format • 38

- field length • 33

- FILLER keyword • 34

- hexadecimal values • 36

- NOSUM • 40

- number of decimals • 34

- numeric values • 35

- REDEFINE • 43

- redefining a field • 34

- REINIT • 37

- report output field size • 38

- RIGHT • 40

- ROUND or ROUNDED • 39

- size/length • 187

- special fields and reserved words • 44

- starting location • 33

- SUM • 40

- VALUE ALL • 36

- VALUE HIGHVALUES • 37

- VALUE LOWVALUES • 37

- VALUE nnn • 35

- VALUE NULL • 36

- WORKAREA • 41

field for MOVE

- ABSOLUTE • 228

- NORMALIZED • 228

- UNSIGNED • 228

field length • 33

field, redefining • 34

FILE • 185, 190, 562

file definitions • 800

file name • 800

file organization • 800, 802

file print

- commands • 475

- dataname • 480

- example • 480

- example of printing in hex and graphics • 476

- filename • 479

- for source • 799

- HEX • 478

- HEXPRINT • 478

- LCPRINT • 478

- LENGTH • 480

- literal • 479

- other print features • 482

- PRINT • 478

- REPORTFILE • 478

- using the OPTION statement • 477

- using the PRINT immediate commands • 478

FILE statement • 101, 729, 738, 748, 800, 801, 802, 815, 816

- BDAM (or DAM) files • 104

- BNF notation • 103

- example • 801

- EXIT and MODIFY • 131

- EXIT files • 105

- I/O area • 134

- IOU-processed files • 136

- ISAM files • 105

- ISAM/VSAM syntax definitions • 125

- MODIFY • 131

- PDS files • 106

- placement • 101

- processing variable-length records • 136

- RETAIN • 131

- sequential files • 107

- SIZE nnnnn • 134

- SSL files • 107

- supported access methods • 103

- syntax definitions • 110

- syntax rules • 103

- unblocked ISAM files in VSE • 140

- VSAM ESDS files • 108

- VSAM KSDS files • 109

- VSAM RRDS files • 109

- when required • 103
- WORK • 135
- files in ISAM, blocked and unblocked • 558
- files, letter • 727
- FILL fillchar • 722
- FILLER • 191
- FINAL, aligning data names and literals at ON • 403
- FINAL, ON • 297, 298, 410
- first byte key • 818
- fixed ESDS VSAM file • 823
- fixed format sequential file • 820, 822
- fixed KSDS VSAM file • 821
- fixed length records • 806
- fixed print position reporting • 406
 - dataname • 406
 - nnn (print location) • 407
- FOOTING statement • 371, 372
 - additional information • 373
 - example • 373, 374
 - FIXED keyword • 372
 - modification statement • 374
 - Tn (1-9) statement • 371
 - WITH n AFTER keyword • 372
- FOR statement • 772, 773
- form letters • 727
- FORMAT letter writing examples • 747
- FROM filename keyword • 707
- FROM statement • 772, 773

G

- G edit code • 314, 739
- general considerations • 336
- GETCOM • 56
- GOTO • 155, 159, 179
- GOTO command • 296
 - ON CHANGE and ON FINAL • 296

H

- H edit code • 315, 740
- help <http://ca.com/support> • 30

HTML

- dynamic allocation of PDS • 367
- dynamic allocation with HFS path • 366
- member names • 360
- report • 358
- using with VISION:Results • 361

HTML document

- using HFS path • 361
- using HTML PDS • 364
- web browser • 369

HTMLTITL dataname • 358

HTMLTITL syntax • 359

I

IDCAMS IBM program • 486, 488

IDCAMS statement • 820, 822

identification of the VISION:Results load library • 799

IDLENGTH (length of identification field) • 731, 748

IDLENGTH nn keyword • 731

IF • 160, 161, 162, 164, 203, 211

- nested • 163
- restrictions • 164

IF command • 144

- class or status tests • 147
- combined range and series compares • 147
- combining compound compares • 152
- compound compares • 150
- dataname • 144, 145
- ENDIF terminates IF statement • 146
- false condition imperatives • 146
- interval selection of data • 154
- literal or dataname • 145
- multiple compares • 150, 153
- random selection of data • 154
- range compares • 146
- SAMPLING command • 154
- true condition imperatives • 145

IF-OR • 204

IF-THEN-ELSE-ENDIF • 163

IMS/DL/I database interface option • 48

INDENT keyword • 730

- example of no values specified • 748

indexed data name • 380

indexes

- arithmetic • 308
 - COMBINE • 308
 - EDIT • 311
 - four types • 306
 - IF • 307, 308
 - INW, INX, INY, INZ • 306
 - MOVE • 306, 308
 - restrictions • 308
- INFCNT data name • 813
- INFILE • 802
- INFILE DD statements • 804
- input data set (INFILE) • 810
- input file definitions • 800
- input line, number of characters per • 59
- Input Output Unlimited (IOU) • 136
- INPUT, ON END OF • 303, 304, 305
- INRECORD data record • 805
- instream to card images • 818
- instream to disk
 - (ISAM, blocked) ISAM file • 817
 - (sequential) • 817
 - (unblocked ISAM with RKP not equal to 0) ISAM file • 818
 - (unblocked ISAM, RKP=0) ISAM file • 818
- instream to printer • 819
- instream to standard label tape • 815
 - (instream data contains JCL) • 816
 - (reformatting instream data) • 816
 - (variable-length records) • 815
- interval selection
 - example • 690, 691, 692
 - interval size • 687, 688
 - of data • 154
 - SAMPLE • 688
 - selection criteria • 688
- IO keyword • 548, 591
- IOAREA keyword (I/O area) • 135
- IOU (Input Output Unlimited) • 136
- ISAM and VSAM files syntax definitions
 - CYLOFL keyword • 130
 - ERASE keyword • 127
 - EXTENTS keyword • 131
 - KEYLEN keyword • 125
 - KEYLOC keyword • 125
 - MASTER keyword • 131
 - PARTKEY keyword • 126
 - POSITION keyword • 126
 - RELBYTE keyword • 129
 - RELBYTEX keyword • 129
 - RETAIN command • 131
 - REUSE command • 127
 - STATUS keyword • 125
 - STATUS return codes listing • 126
- ISAM example • 562
- ISAM file • 801, 810, 811, 814
 - blocked • 805
 - blocked and unblocked files • 558
 - creating a file • 536, 537
 - example • 561, 562, 563, 564
 - generic search • 543
 - ISAM-indexed sequential access method • 535
 - numeric keys • 560
 - packed keys • 560
 - random read • 544
 - random update • 550, 560
 - sequential insert or extend • 554
 - sequential load or extend • 559
 - sequential or skip-sequential read • 558
 - sequential read • 538
 - sequential update • 547
 - skip-sequential read • 540
 - syntax format • 536
 - unblocked • 805, 806
 - unblocked files • 558
 - VSE considerations • 538
 - z/OS considerations • 537
- ISAMFL file • 814
- ISC subparameter on a DLBL statement • 538
- ITERATE • 171
- iterative structures • 167
 - DOUNTIL • 157, 168, 171
 - DOWHILE • 157, 168, 171
 - ENDDO • 169
 - EXITDO • 170
 - ITERATE • 171

J

- J edit code • 315, 740
- JCL

- requirements • 799
- requirements for z/OS utilities • 799
- VM/CMS • 798
- VSE • 797
- z/O • 797

K

- K edit code • 314, 739
- key length coded • 801
- key starts in the first byte • 818
- KEYLEN keyword • 537, 539, 541, 566, 707, 712
- KEYLOC keyword • 537, 539, 707, 712
- keywords
 - ASA • 727, 732, 765
 - COUNT • 567
 - CYLOFL • 130, 538
 - DYLEXT • 730, 731
 - DYLTR • 730, 732
 - DYNAM • 317
 - ENDLTD • 729, 738, 749
 - ENDLTH • 729
 - ENTRY • 707
 - ERASE • 127, 508
 - EXTENTS • 131
 - F entrylen • 706
 - FILLER • 34
 - FIXED • 372
 - FROM filename • 707
 - IDLENGTH • 731, 748
 - INDENT • 730
 - IO • 548, 591
 - IOAREA • 135
 - KEYLEN • 125, 537, 539, 541, 566, 707, 712
 - KEYLOC • 125, 537, 539, 707, 712
 - LASER • 730
 - LEFT • 40
 - LEFTSAVE • 733, 759, 762
 - LENGTH • 707
 - LONG • 730, 753, 759
 - MASTER • 131
 - NOEJECT • 730, 753, 756
 - NOOPTIMIZE • 733
 - NOSUM • 40, 188, 215
 - OPTIMIZE • 733, 746, 747
 - PARTKEY • 126
 - PLUnnn (programmer logical unit) • 369
 - POSITION • 126

- PRIOR • 385
- REDEFINE • 43
- REINIT • 37
- RELBYTE • 129
- RELBYTEX • 129
- RIGHT • 40
- ROUND • 39
- ROUNDED • 39
- STATUS • 125, 706
- SUM • 40, 188, 381
- SUPPRESS • 197, 383
- SWITCH • 733, 762
- tablename • 706
- TALLY • 396
- THRU • 205
- TWOUP • 732, 733, 745, 759, 762
- V maxlen • 706
- WIDE • 730, 759
- WITH n AFTER • 372

- KSIDS
 - fixed format file • 820
 - VSAM file backup to another file • 821

L

- L edit code • 315, 740
- label
 - TAPE options • 121
 - tape to card image • 806
 - tape to disk • 805, 806
 - tape to printer • 806, 807, 808, 809, 810
- LASER keyword • 730
- LASERnn used with AUDPRINT/SYS009 • 732
- LEFT • 40
- LEFTSAVE keyword • 733, 759, 762
- LENGTH keyword • 707
- length of record • 801
- length records, variable • 137, 597, 598
- LENGTH statement • 815
- LETTER command • 729, 738, 742, 749, 755
 - FROM-TO • 742
- letter writing • 727
 - @LTDnn • 734
 - ASA • 727, 732, 765
 - ASA keyword • 732, 765
 - BETWEEN optional factor • 732

BY yyy optional factor • 732
detail statements • 729
DISK optional factor • 732
DYLEXT keyword • 730, 731
DYLEXTxx keyword • 731
DYLTR keyword • 730, 732
DYLTRxx keyword • 732
ENDLTD keyword • 729, 738, 749
ENDLTH keyword • 729
example • 747, 750, 752, 754, 755, 758, 759, 761, 765
exit subroutine • 727
header statement • 729
IDLENGTH keyword • 731, 748
INDENT keyword • 730
laser JCL • 743
LASER keyword • 730
LEFTSAVE keyword • 733, 759, 762
LETTER command • 729, 733, 742, 749, 755
letter files • 727
LONG keyword • 730, 753, 759
LTD definition statement • 729, 732, 733, 738, 742, 746, 748, 755
LTH definition statement • 729, 730, 732, 733, 742, 747, 748
nnnK in DYLEXT keyword • 731
NOEJECT keyword • 730, 753, 756
NOOPTIMIZE keyword • 733
OPTIMIZE keyword • 733, 746, 747
quotation marks use • 738
SWITCH keyword • 733, 762
syntax • 729, 734
SYS number • 732
TAPE optional factor • 732
TWOUP keyword • 732, 733, 745, 759, 762
WIDE keyword • 730, 759
WRITEALL option • 730, 731, 732

letter writing considerations
 @ sign • 744
 @(ASIS) • 744, 745
 @(CSnn) • 736
 @(Eject) • 736
 @(NEWPARA) • 737
 @(SPn) • 735
 @dataname • 734
 @nnn • 737, 745
 ENDLTD • 737
 ENDLTD keyword • 744
 FILE • 738
 LETTER • 738
 literals • 744
 LTD statement • 734, 744
 LTH statement • 738
 quotation marks • 744

letters
 form • 727
 two-up • 732, 759

LETTERxx command • 733

library, SSL (source statement library) • 633

limitations, punctuation • 57

limited output • 813

line analysis, trend • 767, 770

line equation, straight • 768

line spacing • 400
 WITH EJECT AFTER • 401
 WITH EJECT BEFORE • 400
 WITH n AFTER • 401
 WITH n BEFORE • 400

line, number of characters per input • 59

linear regression • 767, 768, 775
 analysis • 771, 775, 777
 best-fitting line • 768
 coefficient of determination • 770, 777
 coefficient of non-determination • 770
 correlation coefficient • 770
 dependent variable • 778
 example • 778, 779
 independent variable • 778
 REGRESSION • 773
 REGRESSION command • 773, 775
 regression line • 769, 776
 slope • 776
 x-axis • 772
 y-axis • 773
 y-intercept • 776

linear regression, syntax • 771, 772, 773
 BY statement • 772, 773
 FOR statement • 772, 773
 FROM statement • 772, 773
 LINEAR statement • 771
 TITLE statement • 772, 780
 XNAME statement • 771
 XSCATTER statement • 771, 772
 YNAME statement • 772
 YSCATTER statement • 773

LINEAR statement • 771

LIST • 188, 191

list members of a partitioned data set • 815

LIST statement • 378

- column heading • 398

- dataname • 380, 387

- dataname AT dataname2 • 391

- dataname AT dataname2 +n • 392

- indexed data names with • 380

- keywords used • 381

- literal use • 393

- minimum requirements for • 379

- placement of list • 379

- PRIOR keyword • 385

- SUM keyword • 381

- SUM1 through SUM6 keywords • 386

- SUPPRESS keyword • 383

- TALLY keyword • 396

- to produce a report • 800

literals • 63, 744

- assigning character values to fields • 65

- assigning values to fields using hexadecimal notation • 69

- assigning values to numeric, packed, or binary fields • 68

- at detail time, aligning data names and • 402

- at ON CHANGE IN, aligning data names and • 403

- at ON FINAL, aligning data names and • 403

- character or alphanumeric • 64

- comparing characters • 65

- comparing numerics • 67

- file or record print • 66

- hexadecimals • 68

- moving characters • 65

- moving hexadecimal • 68

- moving numerics • 67

- numeric literals • 67

- numerics in arithmetic expressions • 68

load library identification • 799

logical unit • 454

LONG • 730, 753, 759

LTD • 729, 733, 738, 748, 755

- first command • 734

- statement • 744

- text of letter • 729

LTH • 729, 730, 732, 733, 738, 742, 747, 748

letter heading • 729

M

macros

- SPIE • 285

- STXIT/PC • 285

maintaining a count of records • 567

MASTER • 538

MATCH

- COMPAREN • 642, 643

- DY282MAT • 637, 642, 644, 645, 646

- DY282MAT invocation of • 664

- DY282MAT match subroutine • 665, 666, 669

- DY282MAT parameters passed to • 672, 673

- DY282MAT replacement routine • 674, 675, 676, 677, 678

- example • 653, 654, 655, 656

- features • 640

- functions • 637

- IF ADVANCED • 646, 671

- IF DUPLICATE • 649, 671

- IF FIRSTDUP • 650, 671

- IF LASTDUP • 651, 671

- IF MATCHED • 647, 648, 671

- MASTERUP • 642, 643

- ON ONE logic • 670

- ORIGINAL • 642, 644

- recommendations and requirements • 642

- SETREAD • 642, 643

- subroutines • 638

- syntax • 641

- user statements • 671

member names, HTML • 360

members of a partitioned data set • 815

MERGE • 657

- DY282MER • 637, 658, 660

- DY282MER parameters passed to • 682

- DY282MER replacement routine • 684, 685, 686

- DY282MER rule for • 659

- error handling • 663

- features • 640

- IF REPLICA • 661, 681, 682

- logic • 637

- merge conditions • 659

- recommendations and requirements • 659

- subroutines • 638

- syntax • 657
- message file for SORT • 800
- methods, access • 567
- mletter writing considerations @(ASIS) • 736
- modification statement, title/footer • 374
- MODIFY • 134
- MODIFY module, writing • 290
 - PARM • 291
 - SIZE • 291
- MODIFY modulename • 132
 - restrictions • 132
- modulename
 - EXIT • 132
 - MODIFY • 132
- MOVE • 217
 - ABSOLUTE fields • 228
 - blanks to a character field • 222
 - character literal to a field • 220
 - CORRESPONDING • 96, 227
 - field or literal to a subtotal field • 222
 - field to a field • 219
 - field to field • 219
 - fields • 219
 - hexadecimal literal to a field • 221
 - IF statements • 217
 - NUMERIC • 222
 - numeric literal to a field • 220
 - OFFSET • 223
- moving to a subtotal field • 222
 - ON CHANGE IN • 222
 - ON FINAL • 222
- multiple compares • 150
 - selecting or processing if all of several conditions are true • 152
 - selecting or processing if any of several conditions are true • 151
- multiple records using IF to select • 211
- multiple reports • 453
 - PICNSAVE • 456
 - REPORTnnn • 453
 - USE • 454
 - VSE example • 467, 472, 473
 - VSE JCL for example • 471, 474
 - z/OS example • 459, 464
 - z/OS JCL example • 463, 466
- multiple requests • 453, 458

- LINEAR • 458
- LTH • 458
- PICNSAVE • 456
- REPORTnnn • 453
- SAMPLE • 458
- SCATTER • 458
- USE • 454

N

- NA edit code • 314
- name, indexed data • 380
- names and literals
 - at detail time, aligning data • 402
 - at ON CHANGE IN, aligning data • 403
 - at ON FINAL, aligning data • 403
- naming conventions • 58
 - data names • 58
 - file names • 59
 - tag names • 59
- NB edit code • 316
- NDA edit code • 315
- NDE edit code • 314
- NE edit code • 313
- NEWPAGE • 159
- NEWPARA • 759
- NEXT • 297
 - ENDIF • 297
- nnnn, DISK • 123
- nnnnn, SIZE • 134
- no INDENT • 748
- NOEJECT • 730, 753, 756
- NOFEED diskette option • 124
- NOOPTIMIZE • 733, 746
- NOQLF • 93, 99
- NOQUALIFIERS • 93, 99
- NORETAIN • 707
- NORMALIZED • 228
- NORMALIZED field • 228
- NOSUM • 40, 188, 214
- NOSUM keyword • 215
- notation
 - Backus-Naur-Form (BNF) • 103

BNF • 73
NP edit code • 316
NU (numeric data) • 34, 127, 232
number of characters per input line • 59
 COLUMNS • 59
numeric data (NU) • 127, 232
NUMFILES
 DISKETTE • 122
 TAPE • 122
NZ edit code • 315

O

OFFSET • 707
ON CHANGE IN • 162, 188, 297, 298, 409
 dataname • 409
 example • 409, 410
ON END OF INPUT • 303, 304, 305
 ACCEPT • 303, 305
 considerations • 305
 example • 304
 ON CHANGE IN • 303, 304, 305
 ON FINAL • 303, 304, 305
 REJECT • 303
 STOP • 305
 SUM • 305
 VSAM • 305
ON END OF SORTING • 244
 example • 245, 246
 ON CHANGE IN • 244
 ON FINAL • 244
ON FINAL • 162, 188, 297, 298, 410
 example • 304, 411
ON ONE • 302, 303
 considerations • 303
 example • 302
 SORT • 303
ONE, ON • 302, 303
ONEBUFF one buffer • 124
operators
 arithmetic • 231
 relational • 144
operators, arithmetic • 231
OPTCD • 549
OPTIMIZE • 733, 746, 747

OPTION • 190
 AllFusion Endeavor environment variables • 83
 allocating memory for tables and arrays • 76
 automatic control totals • 89
 CDLOAD • 73
 COBAPOS • 79
 COBOL II environment • 83
 COBXSIGN • 79
 COLUMNS • 59
 companion product options • 86
 VISION: 60 • 86
 VISION:Excel display • 87
 VISION:Excel keywords • 87
 VISION:Inquiry bridge • 88
 VISION:Report • 88
 VSE spooling VISION:Excel output • 87
CONVENTIONAL • 158
counter display • 81
currency symbol substitution • 79
debugging • 89
 ABEND analysis • 91
 cross reference listings • 90
 expand error message information • 91
 extended error analysis • 92
 generated object code listing • 91
 options useful for Technical Support • 92
 premature sort termination without
 abend • 91
 print entry point address • 91
 verify syntax • 92
default options page display • 89
divide by zero condition • 78
DMAP • 97, 98
DYLTIME format change • 82
environment specification options • 83
extended arithmetic report field display • 81
FILE PRINT • 80
four-digit year support • 79
HEX • 477
HEXPRINT • 477
IBM Language Environment (LE) Support • 83
IF NUMERIC test for PD fields • 75
input columns • 73
input literal data • 74
LCPRINT • 477
match file attributes for VSE sequential files • 78
modifying IF NUMERIC test • 75
NOQLF • 93, 99
NOQUALIFIERS • 93, 99
NULL output file control • 81

- number of lines per page • 89
- old release keyword equivalents • 75
- optimization • 85
- PDS REPLACE • 81
- PRINT • 476, 477
- program input and processing options • 73
- program listing and formatting options • 88
- program output options • 79
- programming mode • 74
- QLF • 41, 93, 99
- qualification of data names • 75
- QUALIFIERS • 99
- REPORTFILE • 477
- resource optimization options • 84
- STRUCTURED • 158, 642, 659
- STRUCTURED2 • 158
- system-determined block size support for z/OS • 82
- user-defined index fields • 77
- USERDEFAULT • 159
- VISION:Results Interface to DB2 • 84
- VISION:Results Interface to DB2 (Dynamic) • 84
- VISION:Results Interface to DB2 (Static) • 84
- VSAM file characteristics • 77
- VSE disk work file • 76
- VSE report file name • 82
- XREF • 97
- XREFA • 97
- OPTION statement • 71, 800
- OR (EXOR), EXCLUSIVE • 226
- order of precedence in overriding specifications • 803
- OUTFILE DD statement • 816
- OUTPUT • 569
- output file definitions • 802
- output file format and length • 802
- output goes to SYSOUT=B • 812
- Output Unlimited (IOU), Input • 136

P

- P edit code • 316, 741
- packed decimal (PD) • 127, 232
- page eject • 742, 756
- PARM • 135

- partial records • 809
- PARTKEY • 495
- PD (packed decimal) • 34, 127, 232
- PDS (partitioned data set)
 - ADD • 617
 - ALIAS • 617
 - CHANGE • 618
 - directory • 632
 - example • 619, 620, 622, 623, 624, 625, 626, 627, 628, 630
 - READDIR • 615
 - READMEM • 615
 - recommendations • 616
 - REPLACE • 618
 - SCRATCH • 619
 - WRITEDIR • 616
 - WRITEMEM • 616
- PDS source definition • 800
- per input line, number of characters • 59
- PERFORM • 155, 161, 299, 300
 - BY • 299
 - considerations • 300
 - example • 299
 - ON CHANGE IN • 162, 300
 - ON FINAL • 162, 300
 - VARYING • 162, 299
- PICNSAVE • 456
 - BY nnnnn • 457
 - idname • 456
 - SYSnnn • 457
 - TAPE • 457
 - temporary file • 456
 - USING • 456
- PL/I • 29
 - example (invoking a PL/I external subroutine) • 288
 - PL1EXIT (an Assembler subroutine) • 287
 - PL1EXIT source program • 289, 290
 - subroutine • 287
 - VSE link edit example • 288
 - z/OS link edit example • 288
- PLUnnn • 369
 - programmer logical unit • 369
- point, exit • 427
- POSITION • 542
- position reporting, fixed print • 406

- preallocated file • 802
- precedence in overriding specifications • 803
- predefined file • 802
- prefixing data names • 344
- print
 - file • 475
 - graphics only • 808, 809, 813
 - hex and graphics • 806, 807, 808, 813, 821, 823
 - sorted VSAM ESDS file • 824
 - sorted VSAM KSDS file • 821
- print commands • 478
 - HEX • 478
 - HEXPRINT • 478
 - LCPRINT • 478
 - PRINT • 478
 - REPORTFILE • 478
- PRINT immediate command • 478
- print position reporting, fixed • 406
- PRINT record number • 482
- printing the instream data file in graphics format • 818
- problem areas • 269
 - VSE • 270
 - z/OS • 270
- procedure logic • 187
- procedure termination • 302
- processing program files • 800
- produce a report • 800
- program execution, directing • 295
- program files being processed • 800
- program statements • 800
- programming tips
 - naming conventions • 58
 - number of characters per line • 59
 - punctuation limitations • 57
- PROTECT diskette option • 124
- punctuation limitations • 57
- PUTCOM • 56

Q

- Q edit code • 316, 741
- QLF • 93, 99

- qualification of data name • 93
 - COUNT • 94
 - DMAP • 97, 98
 - format • 94
 - LENGTH • 94
 - MOVE CORRESPONDING • 96
 - no qualified data names option • 99
 - NOQLF • 93, 99
 - NOQUALIFIERS • 93, 99
 - QLF • 93, 99
 - QUALIFIERS • 99
 - qualifying keywords • 94
 - qualifying multiple reports • 95
 - qualifying work areas • 95
 - XREF • 97
 - XREFA • 97
- QUALIFIERS • 99
- QUIT • 298
- QUITALL • 299
- quotation marks • 744

R

- RANDOM • 127, 566
- random read • 580, 814
 - relative block number • 584
- random selection
 - example • 700, 701, 702
 - IF SAMPLING • 699
 - known universe • 696
 - of data • 154
 - RANDOM • 696, 698
 - reserved words • 701
 - SAMPLE • 696, 697
 - SEED • 697, 698
- random update • 590
 - relative block number • 596
 - relative track number and actual key • 591
 - relative track number and actual record number • 593
- RANDOMX • 127
- range and series compares, combined • 147
- range compares • 146
- RBA • 504
- RBA printing • 823
- RDW • 526, 558

READ • 301

read

- random • 580
- sequential • 586

reblocking of tape • 803

record addressing schemes

- relative block • 567
- relative track and actual record number • 566

record descriptor word • 481

record format • 186, 801

record size • 186, 747

records, variable length • 137, 597, 598

RECSIZE data name • 815

REDEFINE • 43

reformatting instream data • 816

REGRESSION • 773, 775

regression, linear • 767, 768, 775

REINIT • 37, 307

REJECT • 155, 159, 174, 179, 298

relational operators • 144

- EQ • 144
- GE • 144
- GT • 144
- LE • 144
- LT • 144
- NE • 144

RELBLK • 567

RELBYTE • 488, 492, 511, 523

RELBYTEX • 488, 492, 523

RELREC • 489, 493, 512

RELSORT example • 264, 265, 266, 267, 268

RELTRK • 566

reorganize ISAM file • 811

REPLACE • 618

replacing a member in an existing library • 811

REPORT • 353, 427

- aligning data names and literals at detail • 402
- aligning data names and literals at ON CHANGE IN • 403
- aligning data names and literals at ON FINAL • 403
- AND • 370
- ASA • 357
- BETWEEN • 356
- CONTROL • 407
- control break • 407
- example • 370, 412
- exit routines • 427
- fixed print position • 406
- HTML • 358
- line spacing • 400
- LIST • 378
- LONG • 356
- modulename • 357
- ON CHANGE IN • 409
- ON FINAL • 410
- placement • 354
- PLUnnn • 369
- SUBTOTAL • 407
- SYS280Rx • 357
- TITLE and FOOTING statements • 371
- title/footing modification statement • 374
- when required • 354
- WIDE • 355

report tape with ASA carriage control • 810

reporting, fixed print position • 406

REPORTnnn • 453

- PLUnnn • 454
- SYS280Rx • 454

reports suppressing subtotals • 215

reports, multiple • 453

requests, multiple • 453, 458

requirements for utilities JCL • 799

reserved words • 44, 191

- BLANK or BLANKS • 45
- DYLCENTRY1 • 45
- DYLCENTRY2 • 46
- DYLCNAME00 • 46
- DYLCNAME0A-0Z • 46
- DYLCNAME1-99 • 46
- DYLCNxxID • 731
- DYLCNxxLNC • 756
- DYLCNxxSW • 733
- DYLCOMMON • 46
- DYLCOMRG • 47
- DYLCOUNT00 • 47
- DYLCOUNT0A-0Z • 47
- DYLCOUNT1-99 • 47
- DYLDATE • 47

- DYLDATE4 • 47
- DYLDATEPAG • 47
- DYLDATEPG4 • 48
- DYLDLA • 48
- DYLETIME • 48
- DYLEXTxx • 730
- DYLEXTxx keyword • 731
- DYLGREG • 48
- DYLGREG4 • 48
- DYLJULIAN • 49
- DYLJULIAN4 • 49
- DYLLINE • 49
- DYLNOTOT • 49
- DYLNRP • 50
- DYLONE • 50
- DYLPAGE • 50
- DYLPAGEn • 50
- DYLPARM • 50
- DYLPARMLen • 51
- DYLPPAGE • 51
- DYLPRTCOMM • 53
- DYLPRTMAXL • 53
- DYLPRTNUMB • 54
- DYLPRTPAGE • 54
- DYLREPMAXL • 54
- DYLRESET • 54
- DYLRETURN • 55
- DYLSYSDATE • 55
- DYLSYSPARM • 55
- DYLTIME • 55
- DYLTRAN • 56
- DYLTRxx • 730, 732
- DYLTRxx keyword • 732
- DYLUPSI • 56
- FILLER • 191
- GETCOM • 56
- PUTCOM • 56
- SPACE or SPACES • 56
- resource optimization options • 84
- restore a fixed KSDS VSAM file • 820, 822
- RETAIN • 707
- RETAIN|NORETAIN • 722
- RETRIEVE • 709
- retrieve
 - randomly • 814
 - VSAM ESDS file • 824
 - VSAM KSDS file • 821
- RETSORT • 263
 - example • 266, 267, 268
- REUSE • 487, 489
- REUSE option • 822
- RIGHT • 40
- RKP not equal to 0 • 806
- RMODE • 293
- ROUND • 39
- ROUNDED • 39
- rounding • 232
- routine library for SORT • 800
- rules • 341

S

- S edit code • 316
- scatter diagrams • 767
 - BY • 772, 773
 - FOR • 772
 - FOR statement • 773
 - SCATTER nn statement • 771
 - syntax • 771
 - TITLE • 780
 - TITLE statement • 772
 - XNAME statement • 771
 - XSCATTER statement • 772
 - YNAME statement • 772
 - YSCATTER statement • 773
- SCATTER statement • 771
- SEARCH • 710, 711
- SECURE diskette option • 124
- SELECT • 118
- selected records • 809
- selection of data
 - interval • 154
 - random • 154
- selection of particular records • 804
- semicolons (;) used for comments • 57
- SEQUENTIAL • 566
- sequential create
 - BDAM file • 573
 - relative block number • 578
- sequential data set • 804
- sequential read • 586, 814

- relative block number • 589
- relative track and actual record number • 588
- relative track number and actual key • 587

series compares • 147

- combined range • 147

SIZE nnnnn • 134

size, record • 747

SKIP • 128, 541

SKIPx • 128

SORT • 193, 237, 797

- considerations • 271
- example • 245, 246
- ON END OF SORTING • 244
- problem areas • 269
- sorting a dataname...UNTIL • 257
- statement • 800
- VM FILEDEF requirements • 273
- VSE sort considerations • 271
- work file • 800
- z/OS JCL requirements • 271

sort

- VSAM ESDS file in another sequence • 824
- VSAM KSDS file in another sequence • 821

SORT a dataname...UNTIL • 257

- example and flowcharts • 259, 261, 263
- record size • 257
- UNTIL • 258
- UNTIL dataname • 263
- USING • 258

SORT a file...UNTIL

- example and flowcharts • 248, 249, 252, 254
- UNTIL • 247
- UNTIL dataname • 252
- UNTIL filename • 248

sorting a file

- ascending • 242
- descending • 242
- example • 243, 244
- filename • 238
- nK • 239
- USING dataname • 239
- WORK n • 242

sorting an unlabeled file • 801

SORTING, ON END OF • 244

source

- format considerations • 337

listing • 799

- source PDS definition • 800
- Source Statement Library (SSL) • 633
- SPACE or SPACES • 56
- spacing, line • 400
- special fields • 44
 - BLANK or BLANKS • 45
 - DYLCENTRY1 • 45
 - DYLCENTRY2 • 46
 - DYLCNAME00 • 46
 - DYLCNAME0A-0Z • 46
 - DYLCNAME1-99 • 46
 - DYLCOMMON • 46
 - DYLCOMRG • 47
 - DYLCOUNT00 • 47
 - DYLCOUNT0A-0Z • 47
 - DYLCOUNT1-99 • 47
 - DYLDATE • 47
 - DYLDATE4 • 47
 - DYLDATEPAG • 47
 - DYLDATEPG4 • 48
 - DYDLA • 48
 - DYLETIME • 48
 - DYLGREG • 48
 - DYLGREG4 • 48
 - DYljULIAN • 49
 - DYljULIAN4 • 49
 - DYLLINE • 49
 - DYlNOTOT • 49
 - DYlNRP • 50
 - DYlONE • 50
 - DYlPAGE • 50
 - DYlPAGE n • 50
 - DYlPARM • 50
 - DYlPARMLEN • 51
 - DYlPPAGE • 51
 - DYlPRtCOMM • 53
 - DYlPRtMAXL • 53
 - DYlPRtNUMB • 54
 - DYlPRtPAGE • 54
 - DYlREPMAXL • 54
 - DYlRESET • 54
 - DYlRETURN • 55
 - DYlsysDATE • 55
 - DYlsysPARM • 55
 - DYlTIME • 55
 - DYlTRAN • 56
 - DYlUPSI • 56

-
- GETCOM • 56
 - PUTCOM • 56
 - SPACE or SPACES • 56
 - SSL (Source Statement Library) • 633
 - BOOK • 634
 - COUNT • 634
 - example • 635, 636
 - FILE • 633
 - READBOOK • 634
 - STATUS • 634
 - standard error of the estimate • 769
 - standard label tape to
 - card image • 806
 - disk • 804, 805, 806
 - standard label tape to printer
 - (fixed length records, print hex and graphics) • 806
 - (partial records, print graphics only) • 809
 - (report tape with ASA carriage control) • 810
 - (selected records) • 809
 - (undefined length records, print hex and graphics) • 808
 - (variable-length records, print hex and graphics) • 807
 - (variable-spanned records, print graphics only) • 808
 - standard label tape to standard label tape • 802
 - reblocking • 803
 - variable # of input files • 803
 - variable-spanned records with selection • 803
 - starting location • 33
 - statement
 - CONTROL • 194, 195
 - COPY • 208, 209
 - DD • 743
 - FILE • 101, 185, 190
 - IF • 203, 211
 - IF-OR • 204
 - LIST • 188, 191
 - OPTION • 190
 - SORT • 193
 - title • 200
 - title or footing modification • 374
 - statements
 - COPY • 800
 - FILE • 729, 738, 748, 800, 801, 815, 816
 - IDCAMS • 820, 822
 - INFILE DD • 804
 - LENGTH • 815
 - LIST • 800
 - OPTION • 800
 - OUTFILE DD • 816
 - SORT • 800
 - STATUS • 541, 721
 - status coded • 801
 - STATUS keyword • 706
 - status tests • 147
 - MIXED (bits mixed) • 150
 - negative value • 148
 - numeric value • 148
 - ONES (bits on) • 149
 - positive value • 147
 - STOP • 298, 549, 578
 - STOPALL • 298
 - STORE • 708
 - straight line equation • 768
 - STRUCTURED mode • 158
 - structured programming • 155
 - ACCEPT • 155, 159
 - automatic cycle • 171
 - CASE • 156
 - coding standards • 159, 160, 161
 - CONVENTIONAL • 158
 - DUNTIL • 157, 160, 161, 178
 - DOWHILE • 157, 160, 161
 - ELSE • 162
 - ENDIF • 163
 - EXITCASE • 159
 - EXITDO • 159
 - GOTO • 159
 - IF • 162
 - IF-THEN-ELSE-ENDIF • 155, 162
 - invoking • 158
 - ITERATE • 159
 - iteration • 157
 - iterative structures • 167
 - PERFORM • 155, 161
 - recommendations with SORT • 176
 - recommendations without SORT • 174
 - REJECT • 155, 159
 - restrictions • 159
 - selection • 155, 162
 - STRUCTURED • 158
 - STRUCTURED2 • 158
 - techniques • 176
-

- USERDEFAULT • 159
- STRUCTURED2 mode • 158
- subroutine, exit • 727
- SUBTOTAL • 407
- subtotal field, moving to a • 222
- SUM • 40, 188, 381
 - example • 382, 383
- SUPPRESS • 197, 383
 - examples • 384
- suppressing subtotals in reports • 215
- SWITCH • 733, 762
- syntax definitions
 - block size • 114
 - BYPASS • 119
 - CARD/CARDS • 117
 - device specifications • 121
 - DROPERR • 120
 - DUMMY • 119
 - DYNAM • 124
 - file organization • 111
 - filename • 110
 - FROM filename • 112
 - input/output type • 111
 - INTERPRET • 118
 - ISAM/VSAM syntax definitions • 110
 - JCL • 118
 - LENGTH dataname • 115
 - NULL • 120
 - ONEBUFF • 124
 - processing mode • 111
 - record format • 112
 - record size • 113
 - SELECT dataname • 118
 - STATUS dataname • 114
 - SYSnnn • 121
- syntax rules • 57
 - continuation rules • 58
 - literals • 63
 - naming conventions • 58
 - number of characters per input line • 59
 - rules for comparing fields • 60, 61
 - rules for moving fields • 61
 - using punctuation • 57
- SYSIN data stream • 814
- SYSOUT • 743

T

- TABLE • 705
- table keywords
 - ENTRY • 707
 - F entrylen • 706
 - FROM filename • 707
 - LENGTH • 707
 - STATUS • 706
 - tablename • 706
 - V maxlen • 706
- tablename keyword • 706
- tables
 - allocating memory • 705
 - binary search • 712
 - deleting a table • 715
 - error return codes • 717
 - examples • 718, 719, 720
 - fixed-length entries • 709, 710, 713, 714
 - loading • 708
 - logical deletion • 716
 - number allowed • 705
 - physical deletion • 715, 716
 - random replacement • 714
 - random retrieval • 713
 - sequential retrieval from • 709
 - sequential SEARCH • 710
 - sorting • 713
 - space required • 705
 - syntax • 706, 707
 - USE • 717
 - user return codes • 717
 - variable length entries • 708, 711, 714, 715
- tagnames • 59
- TALLY • 396
 - example • 396, 397, 398
- TAPE • 122, 732
- tape
 - label • 804
 - to card image • 806
 - to disk • 805, 806
 - to printer • 806, 807, 808, 809, 810
 - with a standard label • 802
- TAPE LABEL • 121
 - NL • 121
 - SL • 121
 - T • 121

-
- TAPE NUMFILES • 122
 - Technical Support • 30
 - contacting CA • 30
 - <http://ca.com/support> • 30
 - terminate a procedure • 302
 - tests
 - class • 147
 - status • 147
 - THRU • 205
 - time, aligning data names and literals at detail • 402
 - TITLE • 353, 371, 772, 780
 - additional information • 373
 - example • 373, 374
 - FIXED • 372
 - FOOTING • 372
 - modification statement • 374
 - title contents • 371
 - Tn (1-9) • 371
 - WITH n AFTER • 372
 - Title statements (Tn) • 200
 - title/footer modification statement • 374
 - additional information • 376
 - alphanumeric literal • 375
 - data names • 376
 - example • 377
 - limitations and constraints • 377
 - numeric literal • 375
 - reserved words • 376
 - tn (1-9) • 374
 - totals for source • 799
 - TRANSLATE • 309
 - dataname1 • 311
 - dataname1 FROM • 309
 - EBCDIC • 310, 311
 - INTO dataname2 • 310
 - UPPER, LOWER • 310
 - trend line analysis • 767, 770
 - dependent variable • 770
 - example • 780
 - independent variable • 770
 - TITLE statement • 772, 780
 - TREND • 771
 - trend line analysis, syntax • 771
 - BY statement • 772, 773
 - FOR statement • 772, 773
 - FROM statement • 772, 773
 - TITLE • 780
 - TITLE statement • 772
 - TREND nn statement • 771
 - TREND statement • 771
 - XNAME statement • 771
 - XSCATTER statement • 772
 - YNAME statement • 772
 - TREND statement • 771
 - TSIZE • 705, 706
 - TSORT • 713
 - two-up letters
 - two letters are produced side-by-side • 759
 - using the defaults example • 759
 - TWOUP • 732, 733, 759, 762
 - two-up letter processing • 745
- ## U
- U edit code • 315, 740
 - unblocked data with RKP not equal to 0 • 818
 - unblocked data with RKP=0 • 818
 - unblocked ISAM file with RKP=0 • 805
 - unblocked ISAM file, RKP not equal to 0 • 806
 - unblocked OUTFILE when going to SYSOUT=B • 818
 - undefined length records • 808
 - UNIT parameter • 810
 - unit, logical • 454
 - unlabeled tape to standard label tape • 804
 - Unlimited (IOU), Input Output • 136
 - unpredictable results 0C4 ABEND • 301
 - UNSIGNED field • 228
 - update, random • 590
 - USE • 454, 717
 - STATUS dataname • 455
 - use DB2 table definitions • 84
 - USERDEFAULT mode • 159
 - invoking • 185
 - utilities
 - disk (ISAM) to disk (ISAM) • 811
 - disk (ISAM) to standard label tape • 810
 - disk (sequential) to disk • 811
 - disk to card images • 812
-

- disk to disk • 812
- disk to printer • 813
- disk to standard label tape • 810
- instream to card images • 818
- instream to disk • 817
- instream to printer • 819
- instream to standard label tape • 815
- instream to unlabeled tape • 816
- standard label tape to card images • 806
- standard label tape to disk • 804
- standard label tape to printer • 806
- standard label tape to standard label tape • 802
- unlabeled tape to standard label tape • 804
- VSAM
 - define ESDS fixed format file using IDCAMS • 822
 - define KSDS fixed format file using IDCAMS • 820
- VSAM ESDS
 - disk to printer • 823
 - disk to tape • 823
 - tape to disk • 822
- VSAM KSDS
 - disk to printer • 821
 - disk to tape • 821
 - tape to disk • 820

V

- V edit code • 316, 742
- V maxlen keyword • 706

VALUE

- ALL • 36
- character • 35
- hexadecimal • 36
- HIGHVALUES • 37
- LOWVALUES • 37
- null • 36

- variable-length records • 137, 597, 598, 807, 815
 - block size • 525
 - file printing variable files • 526
- ISAM
 - block size • 556
 - file printing variable files • 558
 - length of current record • 557
 - location of data within record • 557
 - output from an input file • 557
 - output from itself • 557
 - record size • 556

- sorting a variable file • 558
 - length of current record • 525
 - output from itself • 525
 - sorting a variable file • 526
- variable-length records VSAM AND ISAM
 - output from an input file • 525
 - record size • 525
- variable-spanned records • 808
- VERIFY diskette option • 124
- VISION:Results Interface to DB2
 - VARCHAR data type support • 348
- VISION:Results load library identification • 799
- VISION:Results work file • 799
- VM • 29
- VM/CMS JCL • 798
- VOL=SER parameter • 810
- VOLVER diskette option • 124
- VSAM • 483
 - creating a file • 485
 - generic search • 496
 - numeric keys • 560
 - packed keys • 560
 - random read • 501
 - sequential insert/extend • 520
 - sequential read • 490
 - sequential update • 507
 - skip sequential read • 493
 - syntax • 484
- VSAM ESDS
 - define fixed format file using IDCAMS • 822
 - disk to printer (fixed ESDS VSAM file, print hex and graphics) • 823
 - disk to printer (retrieve, sort in another sequence, and print) • 824
 - disk to tape (backup a fixed ESDS VSAM file to another file) • 823
 - entry sequence create • 487
 - entry sequence random read • 503
 - entry sequence random update • 516
 - entry sequence sequential read • 491
 - entry sequence sequential update • 509
 - entry sequence skip sequential read • 496
 - examples • 532, 533
 - files • 484
 - key sequence sequential update • 509
 - RELBYTE • 511
 - tape to disk (create/restore a fixed ESDS

- VSAM file from a fixed format sequential file) • 822
- VSAM file • 801
- VSAM file definition • 800
- VSAM KSDS
 - define fixed format file using IDCAMS • 820
 - disk to printer (fixed KSDS VSAM file, print hex and graphics) • 821
 - disk to printer (retrieve, sort in another sequence, and print) • 821
 - disk to tape (backup a fixed KSDS VSAM file to another file) • 821
 - examples • 528, 529, 530, 531
 - key sequence create • 486
 - key sequence random read • 501
 - key sequence sequential update • 508
 - key sequence skip sequential read • 493
 - tape to disk (create/restore a fixed KSDS VSAM file from a fixed format sequential file) • 820
- VSAM RRDS
 - key sequence sequential update • 511
 - relative record create • 489
 - relative record random read • 505
 - relative record sequential read • 492
 - relative record sequential update • 511
 - relative record skip sequential read • 499
- VSE • 29, 732
- VSE JCL • 797
- VSE only in DYLTRxx keyword • 732

W

- W edit code • 316, 741
- WHEN • 165
- WIDE • 730
- WIDE keyword • 759
- word, record descriptor • 481
- WORK • 135
- work areas • 41
- work file for VISION:Results • 799
- WORKAREA
 - data name • 752, 753
 - examples • 41, 43
 - multiple work areas • 43
 - naming of • 41

- QLF • 41
- QUALIFIERS • 41
- statement • 65
- WORKFL file • 805
- WRITE • 301, 549
- write over an existing VSAM file • 820, 822
- WRITEALL • 730
- WRITEALL in DYLEXT keyword • 731
- WRITEALL option in ASA keyword • 732
- writing, letter • 727
- WRKRCD data record • 805

X

- X edit code • 314, 740
- XML • 439
 - benefits • 431
 - creating documents using HFS path • 433
 - creating documents using sequential data sets • 435
 - creating documents using XML PDSE • 434
 - data and schema document generation • 439
 - data document with style sheet, sample • 439, 440, 441
 - data document without style sheet, sample • 439
 - documents on PC web browser • 442
 - dynamic allocation using HFS • 435
 - dynamic allocation using HFS path • 435
 - dynamic allocation using PDSE • 437
 - dynamic allocation using sequential data sets • 438
 - error handling • 447
 - API completion and reason codes • 449
 - WORKAREA fields • 447
 - eXtensible Stylesheet Language Transformation (XSLT) • 442
 - sample program • 439, 445
 - sample XSL style sheets • 442
 - schema document, sample • 441
 - support • 431
 - using • 431
 - XML data document without style sheet • 444
- XMLFILE
 - ddname • 431
 - syntax • 431
- XMLWRITE ddname • 432

XNAME statement • 771

XREF • 97

XREFA • 97

XSCATTER statement • 772

Y

Y edit code • 313, 739

YNAME statement • 772

YSCATTER statement • 773

Z

Z edit code • 315, 740

z/OS • 29

z/OS JCL • 797

z/OS utilities • 799