Advantage™ VISION:Report®
Advantage™ VISION:Forms™

Reference Guide

16.1

Computer Associates™

RPREF161.PDF/D21-002-010

Contents

Chapter 1: Introduction

Chapter 2: File Specifications and Data Definitions

Chapter 3: Statement Format

Chapter 4: Examples

Chapter 5: Troubleshooting and Memory Requirements

Chapter 6: Optional Material

Chapter 7: QUIKVSAM

Appendix A: Invoking VISION:Report from VISION:Results or VISION:Eighty

Index

Chapter

1

Introduction

VISION:Report® is an easy-to-use program development and report writing tool that you use to build and execute programs, minimizing the amount of programming time and effort required to fix a file, prepare a report, or generate test data.

There is virtually no limit to the types of reports VISION:Report can create. Preprinted forms, memos, letters, and labels are handled with ease. Arithmetic operations are provided with automatic decimal alignment and rounding, but you can override these features at any time. You can produce up to eight reports in one program. VISION:Report also supports European format for numeric printing.

VISION:Report uses a straightforward, self-documenting, COBOL-like language. First time users can be up and running after less than eight hours of training. In addition, because VISION:Report requires you to define only those fields to be used, time-consuming COBOL file definition activity is eliminated.

VISION:Report also allows you to save file definitions and program source statements in CA-Librarian®, CA-Panvalet®, PDS, or other source statement libraries to be shared by other programs, avoiding duplicate programming work. Character, zoned, decimal, packed, binary, and unconventional data types are supported. In addition, you can validate programs prior to their execution.

One common language means that VISION:Report is easier to learn and manage than several individually sold utilities. You can accomplish everything from file comparison and format conversions to testing, debugging, and prototyping using VISION:Report.

Using VISION:Report, you have the option to access, retrieve, update, and report on data from VSAM, sequential, and optionally:

- Computer Associates CA-IDMS/DB®
- IBM®
- DB2® and SQL/DS
- IMS and DL/I

Also, a CALL verb is available to access callable databases. To enhance file access, VISION:Report:

- Internally reads OS/390®-partitioned data sets, CA-Librarian and CA-Panvalet libraries.
- Provides a CALL command for invoking subroutines written in Assembler or COBOL.
- Allows literals to be used as parameters in a CALL statement.

VISION:Report performs multiple functions with just one pass of the database. Conditional record selection, sorting, translating, matching, and merging can all occur with only one read of the master file. For further efficiency, VISION:Report also:

- Handles internal sorts in one statement.
- Releases and returns records from SORT in one easy step.
- Accepts compound conditional IF statements, allowing for complex data selection.
- Allows conditional branching.

VISION:Report operates under z/OS, OS/390, MVS/SP/XA/ESA, VSE/SP/ESA, and VM/CMS.

Note: Throughout this document, MVS is synonymous with z/OS and OS/390, unless specified differently.

About This Guide

This guide covers the fundamental principles of VISION:Report.

Chapter	Description
1. Introduction	Contains an introduction to VISION:Report and describes this guide.
2. File Specifications and Data Definitions	Describes file specifications and data definitions, including literals, data areas, file availability, field definitions, named accumulators, VSAM support, VSE and MVS considerations, database files, and user abend codes.
3. Statement Format	Describes the syntax for all of the VISION:Report commands and keywords.

Chapter	Description
4. Examples	Contains extensive examples of VISION:Report features.
5. Troubleshooting and Memory Requirements	Describes troubleshooting tips, memory requirements, and debugging facilities. This chapter also contains information about program checks and how to report a problem to Technical Support.
6. Optional Material	Describes the VISION:Report optional features, including a random number generator, date and time subroutines, table editors, a multiple report processor, and various subroutines and interfaces.
7. QUIKVSAM	Describes QUIKVSAM, which provides you with native mode access to VSAM data sets (files), enabling you to support commonly used VSAM functions with the minimum of effort.
Appendix A, Invoking VISION:Report from VISION:Results or VISION:Eighty	Describes how to invoke stand-alone VISION:Report applications and VISION:Report applications that access IMS or CA-IDMS/DB databases from your VISION:Results™ or VISION:Eighty™ program.

Contacting Computer Associates

For technical assistance with this product, contact Computer Associates Technical Support on the Internet at http://esupport.ca.com/. Technical support is available 24 hours a day, 7 days a week.

Chapter 2

File Specifications and Data Definitions

Statement Format and Sequence

The following VSE file specification formats are fixed form. Layout specifications are included in this section for each statement.

File Specification Format	Description
INF	Define first input file
DET	Define second input file
INC	Define third input file
IND	Define fourth input file
INA, INB, INE, ING - INZ	Define additional input files
OFA - OFZ	Define output files

All other VISION:Report statements are free-form within the following rules:

- Sequence Numbers

  Positions 1-3 or 1-4 must contain a sequence number when the option SEQCHK=YES is in effect (see the section OPTION, in Chapter 3). If the SEQCHK=NO option is in effect, a sequence number is not required except on statements that are the subjects of the GOTO and PERFORM transfer statements. The SEQCHK=NO option allows you to start a statement anywhere from column 1 to 30, except for HDR which must start in column 5.

- When the SEQCHK=NO option is in effect, VISION:Report assumes all statements without sequence numbers have a sequence number of 000. Therefore, a GOTO 000 produces unpredictable results.

- A statement may contain only a sequence number, without anything else coded on the statement, except that the sequence number of the EXIT statement must be on the same card.

Another available option is STMTEND=nn. This allows sequence numbering on the right side of all statements. It does not eliminate the need for VISION:Report sequence numbers on the left side of the statement. See the section OPTION, in Chapter 3 for further explanation.

- All words following the statement are free-form, but must be present and in the correct sequence. Optional words such as NOT and OR in the IF statement are specifically identified as optional.
- Only one verb per statement is allowed and each statement must be completely contained on one line, except as noted below. The following sequence is valid:

```
020 IF INF7-10-P IS NUMERIC
030 GO TO 060.
```

The following statement is invalid.

```
020 IF INF7-10-P IS NUMERIC GO TO 060.
```

You can continue these verbs on additional lines without the use of a non-blank character in the continuation column.

VERBS		
ACCUM	HEXEXPD	OR
ADD	IF	PRINT
AND	MOVCOND	REPORT
BREAK	MOVE	SORT
CALL	MOVEXPD	SUB
CONDATE	MOVNUM	TRAN
DIVD	MOVZON	TRNT
EXDATE	MSHIFT	XOR
HEXCOND	MULT	

- Blank lines are permitted in order to separate statements into logical groups such as: equates, declaratives or executables.
- Sequence numbers (or labels) serve as transfer points. A label can appear on a line by itself, except when you use it to identify the EXIT verb. For example:

```
PERFORM 30 THRU 40
             .
             .
             .

30                                  /* No verb on this line
   MOVE ABC TO DEF                  /* This will be performed first
   MOVE GHI TO JKL                  /* This will be performed second
40 EXIT                             /* Label must be on same line
```

- You can insert comments in several ways:
  - Use sequence numbers (or labels) followed by an asterisk and the comments on the rest of the statement. SEQCHK option is immaterial.

    You can place an asterisk in any position; comments then follow.
  - After the statement, leave at least one space followed by '/* '; the rest of the statement is a comment.

The STMTEND option dictates the end of one statement and if there is a continuation statement. For simplicity, end a comment before the specified STMTEND.

Following are some examples:

```
1*    THIS IS A COMMENT
30 /* THIS IS A COMMENT
100*  THIS IS A COMMENT
120 * THIS IS A COMMENT
1000* THIS IS A COMMENT
1050 *THIS IS A COMMENT
 *THIS IS ANOTHER COMMENT
  *   THIS IS ANOTHER COMMENT
   *  THIS IS ANOTHER COMMENT
  /*  THIS IS ANOTHER COMMENT
 /*   THIS IS ANOTHER COMMENT
1100  GET INF                /* COMMENT ON A LINE
1200  MOVE A TO B/* THIS IS AN INVALID STATEMENT
```

Literals

Literals are valid for most statements. See each verb for the valid forms. You can use embedded quotation marks in character literals. C'"180"' generates '180'.

Data Areas

All data fields have a maximum of 15 digits. For example, specify a field as WST000000000000001-000002147483648, giving you a wide range of addressability. The practical limit is 10 digits because the largest size of contiguous storage obtained from the operating system is 2G.

The practical limit for input and output file areas is 5 digits because this contains the maximum record size recognized by the operating system.

Input data is available in fixed areas and is written from fixed areas. These areas are customized to the appropriate length based on record lengths supplied in the INx, DET, and OFx statements (VSE) or based on the record lengths obtained from the data sets and/or DD statements (MVS). The following areas are available to you:

Area	Description
CTA-CTP	Sixteen accumulators (each is 8 bytes, packed decimal and is initialized to zeros). There are ten sets of the accumulators — one set for each of the nine possible total levels, plus one set for grand totals. Each accumulator is addressable.
DET	A GET DET statement causes the next logical record of the second input file to be made available to you in this area.
FUN	A 100-byte function area. If you code an IF...ONTABLE statement, the found function is placed in this area when a match is made against the table. FUN contains the function portion of the table entry only. It does not contain the argument.
HDx	Printer header areas (HDA through HDF). These areas are used to store the HDR statements.
INA-INZ DET	A GET INx statement causes the next logical record of the current input file to be made available to you in this area. Note that you do not need to specify the input files in alphabetical order (that is, INA does not have to be specified before IND, and so forth).
LCT	A 2-byte packed number of the print lines available for use on the current page.
OFA-OFZ	Output area for the OFx output file. Assemble the output record in this area by using MOVE verbs prior to issuing a WRITE OFx statement. In VSE, the contents of the OFx areas are not available for use after a WRITE statement is issued. A form of PUT LOCATE mode output is used; a WRITE statement causes the output pointer to be advanced to the next record and/or to the next block. In MVS, the output area is still available after the WRITE statement. PUT MOVE processing is used. Note that you do not need to specify the output files in alphabetical order (that is, OFA does not have to be specified before OFD, and so forth).
PCB	IMS-DL/I PCB. When used by IMS-DL/I at entry to VISION:Report, PCB is initialized with the pointer (or address) to the first PCB.

Area	Description
PNR	A 4-byte packed number of the page number currently being printed. This is also referred to in an HDR statement as PG.
PRT	A position area, limited by the OPTION PRTSIZE, that is written to the printer by the symbolic unit SYSLST (VSE) or ddname SYSPRINT (MVS).
PTA-PTH PTR	Generalized pointers for variable address indexing.
PUN	An 80-position area that is written to the punch by the symbolic unit SYSPCH (VSE) or ddname SYSPUNCH (MVS).
SAV	An area generally used for working storage. The size of this area is specified by the value of SAVAREA (see the section OPTION, in Chapter 3). VISION:Report initializes this area to binary zeroes (HEX 00).
TBH	Table hit address.
TSA	Table start address. On all subsequent calls to QUIKIMS, it will contain the pointer to the PCBnn requested in the QUIKIMS call.
VAL	A value communications area. See VAL Area. Briefly, the VAL area is where the IPL date and its variation are made available to you. UPSI bit settings are expanded and presented in a F1 (on) and F0 (off) manner for VSE users. Within the VAL area, VISION:Report provides various combinations such as the IPL date, time of day, PARM date and return codes.
WST	An area of memory available to you for working storage. This area can be used to contain such items as, indicators, accumulators and working areas. The size of this area is determined by the value of WSTSIZE (see the section OPTION, in Chapter 3). VISION:Report initializes this area to binary zeros (hex 00).

All references to data fields are made by coding an AREA / POSITIONS description that comprises a field definition. (See Field Definition.)

VAL Area

The value communications area contains the following values. Access is by the VAL area (such as, VAL1-4-P). (See the section EQU, in Chapter 3 for default equates to the VAL area.)

Position	Format	Contents
1-4	Packed Decimal	Length of last record read from an undefined file (UNDEF). You may create or modify this value for output to UNDEF files. On WRITE to UNDEF files, you must define the desired block size here. If LRECL=X, this field, if output, is binary.
5-12	EBCDIC	IPL date as obtained from the VSE communication region: DD/MM/YY or MM/DD/YY. If there is a // DATE JCL statement, the date will not be the IPL date, but the date on the JCL statement. At end of job (/&), the system will automatically reset the date back to the IPL date.
13-21	EBCDIC	IPL month in full English spelling.
22-29	EBCDIC	The 8 VSE UPSI bits expanded to 1 byte per bit. A binary 0 bit converts to X'F0'; a binary 1 bit converts to X'F1'. In the MVS version, this field contains the first 8 bytes of PARM=information from the EXEC statement. If less than 8 bytes are present, the remainder of this field contains EBCDIC zeros.
30-37	Packed Decimal	Divide remainder after execution of DIVD statements.
38-45	EBCDIC	Job name from JOB statement.
46-49	EBCDIC	Step return code at EOJ (initialized to zeros). Also, this is the return code from the last CALL made.
50-55	EBCDIC	Date, MMDDYY.
56-61	EBCDIC	Date, YYMMDD.
62-66	EBCDIC	Time, HH:MM
67-70	EBCDIC	User ABEND code used with ABEND verb (initialized to 4095).

Position	Format	Contents
71-75	EBCDIC	Julian date YYDDD. In VSE, the date is the IPL date obtained from the VSE Communications Region and also reflects a date changed by any / / DATE JCL statement.
Positions 76-179 for MVS contain the following:		
76-77	Binary	Byte count contained in the data portion of the PARM= parameter of the EXEC statement.
78-177	EBCDIC	As many bytes of PARM information as specified in positions 76-77 (up to 100 bytes).
178-179		Reserved.
Positions 76-179 for VSE contain the following:		
Note: See the section OPTION, in Chapter 3, PARMEXE keyword. If PARMEXE = YES, VAL 76-179 is the same as for MVS above.		
76-77		Reserved.
78-88	EBCDIC	VSE COMREG user area. (11 bytes from positions 12-22 of the partition communication region.) These contents are also in VAL 261-271, regardless of option.
89-179		Reserved.
Positions 180-239 for VSE and MVS contain the following:		
180	EBCDIC	The level number of the present break or totals. F1 through F9 for levels one through nine and F (X'C6') for the final totals.
181	EBCDIC	This position serves as an indication of whether or not you want to suppress spacing and printing for a given level of totals. VISION:Report initializes to a space (X'40'). You should place an N (X'D5') here to cause VISION:Report to suppress spacing, suppress moving the totals for current level to print, and suppress printing current total levels. The totals for a given level are rolled to the next level and the present level is zeroed regardless of this indicator's setting.

Position	Format	Contents
182	EBCDIC	This position serves as an indicator for page eject. For example, if you want to do something with the accumulators for a given level other than print (such as, write to tape, disk), or you may wish to force a new set of headings on the printer. VISION:Report initializes to a space (X'40'). If you place an E (X'C5') here, VISION:Report forces a new set of headings before processing the next total level and/or exiting from the totals processing section.
183-195		Reserved.
196	EBCDIC	Set to EBCDIC E (X'C5') when INF reaches physical end of file.
197	EBCDIC	Set to EBCDIC E (X'C5') when DET reaches physical end of file.
198	EBCDIC	Set to EBCDIC E (X'C5') when INC reaches physical end of file.
199	EBCDIC	Set to EBCDIC E (X'C5') when IND reaches physical end of file. For other input files, see VAL Area, VAL415 and up.
200	EBCDIC	Set to EBCDIC E (X'C5') when SORT EOF (last record) has been resumed using the SORT-AREA-RETURN feature.
201-204	Binary	You must place record length here (when using SORT with Area-V option) prior to RELEASE.
205-208	Binary	Contains the length (bytes) of the last phase/module called or loaded by you.
209-222		Reserved.
223	Binary	Contains the condition code returned from bits 18-19 of the PSW following an AND, OR, or XOR operation.
224	Binary	Contains the non-zero function byte returned following a TRNT operation.
225-228	Binary	The number of bytes scanned on a true condition in a WHEN or TRNT statement.

Position	Format	Contents
229-231	EBCDIC	Contains the name of the last file on which an I/O operation was performed.
232-235	Binary	Contains the record count of the last file on which an I/O operation was performed.
236-239	Binary	Contains the DTF or DCB memory address of the last file on which an I/O operation was performed.
Positions 240-259 contain feedback information for the last VSAM statement processed:		
240	EBCDIC	VSAM file type: K = KSDS L = LDS E = ESDS R = RRDS
241	EBCDIC	VSAM access type: C = CLUSTER P = PATH X = ALTERNATE INDEX
242		Reserved.
243-246	Binary	Record length of the record last retrieved or updated.
247	Binary	VSAM return code (RC).
248	Binary	VSAM error code (EC).

Position	Format	Contents
If Option VSAMER = NO, Positions 249-260 are defined as follows:		
249-252	Binary	VSAM RBA (relative byte address) of record last retrieved or updated.
253-255	EBCDIC	VSAM error word. Contains certain VISION:Report VSAM errors (RC,EC). Contents are as follows: OK Last VSAM statement processed OK RNF Record not found DUP Duplicate record SEQ Record out of sequence EOF End of file ERR RC/EC is not VISION:Report decodable
256-260		Reserved.
If Option VSAMER = YES, Positions 249-260 are defined as follows:		
249-256	Binary	VSAM XRBA (relative byte address) of record last retrieved or updated.
257-259	EBCDIC	VSAM error word. Contains certain VISION:Report VSAM errors (RC,EC). Contents are as follows: OK Last VSAM statement processed OK RNF Record not found DUP Duplicate record SEQ Record out of sequence EOF End of file ERR RC/EC is not VISION:Report decodable
260		Reserved.
261-271	EBCDIC	VSE COMREG user area. (11 bytes from positions 12-22 of the partition communication region.) Same as VAL78-88. See the section OPTION, in Chapter 3, PARMEXE keyword.
272		Reserved
273-280	EBCDIC	Constant of 'SQLCA '.

Position	Format	Contents
281-284	Binary	Contains length of SQLCA (136).
285-288	Binary	DB2 or SQL/DS Return Code ■ If 0, successful execution, although there might have been warning messages. ■ If positive, successful execution but with an exception condition. ■ If negative, error condition.
289-290	Binary	Tokenized error message; length range 0-70.
291-360	EBCDIC	Tokenized error message; contains one or more tokens, separated by X'FF'. Length of this field is variable, with a maximum of 70.
361-368	EBCDIC	Product signature; if error, diagnostic information such as the name of the module that detected the error. In all cases, the first three characters are DSN for DB2 or ARI for SQL/DS.
369-372	Binary	Internal Relational Data System Code.
373-376	Binary	Internal Data Manager Error Code.
377-380	Binary	Number of rows altered by insert/update/delete.
381-384	Binary	Diagnostic information.
385-388	Binary	Diagnostic information.
389-392	Binary	Diagnostic information.

The warning flag indicators below contain a blank if no error condition exists; otherwise, they contain a "W" to indicate a warning or an error has occurred.

Position	Format	Contents
393	EBCDIC	Warning Flag Indicator 0; "W" if any other warning flags (1-7) contain a 'W'.
394	EBCDIC	Warning Flag Indicator 1; "W" if value of a string column was truncated when assigned to a host variable.
395	EBCDIC	Warning Flag Indicator 2; "W" if null values were eliminated from the argument of a column function.

Position	Format	Contents
396	EBCDIC	Warning Flag Indicator 3; "W" if the number of columns is larger than the number of host variables.
397	EBCDIC	Warning Flag Indicator 4; "W" if a prepared UPDATE or DELETE statement does not include a WHERE clause.
398	EBCDIC	Warning Flag Indicator 5; "W" if the SQL statement was not executed because it is not a valid SQL/DS statement.
399	EBCDIC	Warning Flag Indicator 6; "W" if addition of a month or year duration to a DATE or TIMESTAMP results in an invalid day.
400	EBCDIC	Warning Flag Indicator 7; "W" if one or more non-zero digits were eliminated from the fractional part of a number used as the operand of a multiply or divide.
401	EBCDIC	Warning Flag Indicator 8; "W" if a character that could not be converted was replaced with a substitute character.
402	EBCDIC	Warning Flag Indicator 9; "W" if arithmetic exceptions were ignored during COUNT DISTINCT processing.
403	EBCDIC	Warning Flag Indicator A; "W" if a character conversion error occurred while filling the SQLCA. Indicates that at least one character field contains an invalid code point. If all character fields are valid, this is blank.
404-408	EBCDIC	Return code for the outcome of the most recent execution of an SQL statement. The range is 00000 through 65535.

Position	Format	Contents
409-410	Binary	Last DB2 or SQL/DS function when error occurred. 0 - Execute Immediate 1 - Prepare 2 - Open 3 - Fetch 4 - Close 5 - Execute 6 - Update Current 7 - Delete Current 8 - Commit Work 9 - Rollback Work 10 - Connect TO 11 - Connect 12 - Set CURRENT PACKAGESET 13 - Set SQLID = :host 14 - Set :host = CURRENT TIMEZONE 15 - Set :host = CURRENT TIMESTAMP 16 - Set :host = CURRENT TIME 17 - Set :host = SQLID 18 - Set :host = CURRENT SERVER 19 - Set :host = CURRENT PACKAGESET 20 - Set :host = CURRENT DATE
411-414	EBCDIC	4-digit year (YYYY)
415	EBCDIC	Set to EBCDIC E (X'C5') when INA reaches physical end of file.
416	EBCDIC	Set to EBCDIC E (X'C5') when INB reaches physical end of file.
417	EBCDIC	Set to EBCDIC E (X'C5') when INE reaches physical end of file.
418	EBCDIC	Set to EBCDIC E (X'C5') when ING reaches physical end of file.
419	EBCDIC	Set to EBCDIC E (X'C5') when INH reaches physical end of file.
420	EBCDIC	Set to EBCDIC E (X'C5') when INI reaches physical end of file.
421	EBCDIC	Set to EBCDIC E (X'C5') when INJ reaches physical end of file.
422	EBCDIC	Set to EBCDIC E (X'C5') when INK reaches physical end of file.

Position	Format	Contents
423	EBCDIC	Set to EBCDIC E (X'C5') when INL reaches physical end of file.
424	EBCDIC	Set to EBCDIC E (X'C5') when INM reaches physical end of file.
425	EBCDIC	Set to EBCDIC E (X'C5') when INN reaches physical end of file.
426	EBCDIC	Set to EBCDIC E (X'C5') when INO reaches physical end of file.
427	EBCDIC	Set to EBCDIC E (X'C5') when INP reaches physical end of file.
428	EBCDIC	Set to EBCDIC E (X'C5') when INQ reaches physical end of file.
429	EBCDIC	Set to EBCDIC E (X'C5') when INR reaches physical end of file.
430	EBCDIC	Set to EBCDIC E (X'C5') when INS reaches physical end of file.
431	EBCDIC	Set to EBCDIC E (X'C5') when INT reaches physical end of file.
432	EBCDIC	Set to EBCDIC E (X'C5') when INU reaches physical end of file.
433	EBCDIC	Set to EBCDIC E (X'C5') when INV reaches physical end of file.
434	EBCDIC	Set to EBCDIC E (X'C5') when INW reaches physical end of file.
435	EBCDIC	Set to EBCDIC E (X'C5') when INX reaches physical end of file.
436	EBCDIC	Set to EBCDIC E (X'C5') when INY reaches physical end of file.
437	EBCDIC	Set to EBCDIC E (X'C5') when INZ reaches physical end of file.
438-439	Binary	Length of SQL message.
440-759	EBCDIC	SQL error messages 1-4, each error message being 80 bytes.
760-799	Reserved	Reserved for future expansion.

File Availability

Term	Definition
Input	Twenty-seven input files are allowed by way of DET and INA through INZ. In addition, VISION:Report allows one table file to be loaded. If a CALL such as QUIKIDMS (IDMS interface), QUIKIMS (IMS interface), QUIKDLI (DL/I interface), QUIKVSAM (VSAM), or a user-written CALL is being used, there are no real limits to the number of input files.
Output	Twenty-six output files are allowed by way of OFA through OFZ. In addition, one printer file and one punch file are allowed by PRT and PUN. If QUIKVSAM, user CALL, QUIKIMS, QUIKIDMS, or QUIKDLI, and such are used. There are no real limits to the number of output files.
Number of Files	The total number of files is limited to the I/O capacity, which is 27 for input files and 26 for output files. This total includes those files used in QUIKVSAM, as QUIKVSAM uses the same table space for I/O capacity. However, callable user written subroutines that invoke files or databases are not included in the I/O capacity of 53 input and output files.
Operator or System Console	Data may be accepted from the console or displayed on the console.
VSE	Files may reside on tape (fixed, variable, or undefined) or disk (fixed or variable). VSE VISION:Report allows VSAM or ISAM files to be created or loaded. Generic key capability exists for VSAM or ISAM and updating is supported. FBA disk files and 9340/9345, 3390, 3380, and older disk devices are supported.
Database	VISION:Report supports the following databases by using special interfaces: DB2® or SQL/DS™, DL/I®, IMS, CA-IDMS/DB. VISION:Report also supports ADABAS®, TOTAL®, and most other commercial databases through CALL interfaces normally provided by the vendor.

Field Definition

A VISION:Report field definition is defined instream and is the equivalent of a COBOL data name. A field definition consists of a concatenation of the area name that contains the field and the FROM-TO positions of the field within that area. A packed decimal field is indicated by the suffix -P, while a binary field is indicated by the suffix –B, and a zoned decimal field is indicated by the suffix –Z.

The equated data area (or fieldname) length has been expanded from 14 characters to 34 characters. This provides closer conformity to COBOL standards as well as allow some extra characters, such as a dash (-), to be used as separators.

Field Definition Examples

Example	Description
INF7-10	Positions 7 through 10 of the INF input record.
DET7-10	Positions 7 through 10 of the DET input record.
OFK9940-10039	Positions 9940 through 10039 of the OFK output record.
INZ11001	Position 11001 of the INZ input record.
INF11-13-P	A packed field in positions 11 through 13 of the INF input record.
UNIT-PRICE	An equated data area—see the section EQU, in Chapter 3.
ARFILE-INPUT-UNIT-PRICE	An equated data area—see the section EQU, in Chapter 3.
PNR1-4-P	Page number field.
AMOUNT	An equated data area—see the section EQU, in Chapter 3.
PRT76-79	Printer positions 76-79.
PRT76 2C	Definition of the starting or the leftmost position to edit a field in the printer area. The 2C produces an edit mask with 2 digits to the right of the decimal and commas inserted where appropriate.

Example	Description
PRT76 2E	Same as PRT76 2C, except that European-type punctuation is generated. Commas are used to denote decimal positions and periods are used to separate thousands.
WST1-4-B	Positions 1 through 4 of working storage, binary format.
WST1-4	Positions 1 through 4 of working storage, EBCDIC format.
WST9500-10500	Positions 9500 through 10500 of working storage, EBCDIC format.
TSA1-65	Positions 1 through 65 of table entry currently pointed to by the TSA (table start address) pointer.
PTA5-7-P	Positions 5 through 7 (packed decimal) currently pointed to by the generalized pointer PTA.
WST3-P ZEROES	Position 3 of working storage, packed format, initialized to zeros.

Note: Only significant digits must be coded and a one-position field requires no TO element.

VISION:Report can accept values up to 15 characters long in field definitions. However, files (DET, INA-INZ, and OFA-OFZ) are limited to a system-restricted 32K (5 digits) maximum, and storage areas (SAV,WST) are limited to a system-restricted 2G (10 digits) maximum. For example, you can directly specify:

```
MOVE DET9000-12000 TOOFA25000
EQU  AR-LAST-NAME  WST2115483633-2115483747
```

Leading zeros are not required, but will be accepted if present.

Field Definitions and Sizes

Field Type	Short	Length	Range	Comments
Character	C	1-nnnnnnnnnn	characters	Up to 2 gigabytes addressable
Zoned	Z	1-15	Digits	Zoned Decimal
Packed	P	1-10	Digits	Packed Decimal
Binary	B	1-8	Hex Values	Signed

Named Accumulators CTA Through CTP

There are ten sets of accumulators — one set for each of the nine possible total levels, plus one set for grand totals.

VISION:Report has 16 accumulators, each consisting of 8 bytes, packed decimal. All 160 accumulators are initialized to packed zeros.

Each accumulator is addressable.

```
Accumulator one is CTA.
Accumulator two is CTB.
Accumulator three is CTC.
        .
        .
        .
Accumulator sixteen is CTP.
```

Each set of the accumulators is prefixed with 26 bytes of VISION:Report internal control information.

CONTROL	CTA	CTB	CTC	CTD	CTE	CTF	CTG	CTH	CTI	CTJ	CTK	CTL	CTM	CTN	CTO	CTP
1																
2																
3																
4																
5																

CONTROL	CTA	CTB	CTC	CTD	CTE	CTF	CTG	CTH	CTI	CTJ	CTK	CTL	CTM	CTN	CTO	CTP
6																
7																
8																
9																
10																

The previous described memory is contiguous from the control information for level one through the last or eighth byte of CTP of the grand or final totals.

The accumulators and their control information occupy 1540 memory positions — ((16 * 8) + 26) * 10.

The accumulators' addresses are set to point to level one or the minor level accumulators at all times when control is outside your total level subroutine.

Any VISION:Report verb may reference the accumulators. But, VISION:Report treats the individual accumulators as 8 byte, packed decimal fields for operations such as rolling, clearing and editing to the printer.

To direct VISION:Report to use less than 8 bytes of an accumulator, always use the rightmost positions, such as CTA4-8-P and CTA7-8-P.

If the furthermost left positions are used, a data exception can occur and/or there could be no sign in the units position. The sign is treated as data and is invalid. In addition, the magnitude of the data will be out of context.

VSAM Support

VSAM functions are supported with calls to QUIKVSAM (see for further details) or with native VSAM functions.

QUIKVSAM and native VSAM functions can now be coded together in the same program.

Using native VSAM functions, you can read sequentially, read randomly, add, change, or delete VSAM records using VISION:Report statements. The following is a list of verbs that provide VSAM support:

VERBS		
OPEN	CLOSE	GET
WRITE	REWRITE	SETGENKEY
READ	DELETE	ADDRECORD
TCLOSE	CLOSER	

The following are prerequisites to using any of the above verbs on VSAM files:

- A system generated with VSAM support.
- VSAM and AMS modules in the appropriate library.
- A master and/or user catalog.
- Any VSAM file (cluster, path, or alternate index) referenced must have been defined using AMS (IDCAMS).
- A partition/region size large enough to accommodate VISION:Report and VSAM routines (usually 512K or larger).

Although VISION:Report does extensive validation, it is not possible to validate everything. You must be responsible for such items as the data integrity, valid record lengths and inclusion of keys where required. Any and all violations detected by VSAM have the appropriate return/error codes (RC/EC) set in VAL247-248. Do not check the RC/EC together for zeros, but individually, as a non-zero return code may not necessarily be invalid, but only informational in nature. VISION:Report handles the errors in one of the following ways:

If the ONERROR operand was not used:

- The error is not VISION:Report decodable. An error message is printed and the job is canceled. You should refer to the appropriate manuals for explanation of the error.
- The decoded error word is placed in VAL253-255 or VAL257-259 and execution continues with the next statement. It is your responsibility to check for any and all errors.

If the ONERROR operand was used:

- The VSAM error word in VAL253-255 or VAL257-259 is set to either the decoded word or ERR when the error is not VISION:Report decodable. It is your responsibility to check for any errors. The VISION:Report decodable errors for VAL253-255 or VAL257-259 are described in VAL Area.

VSAM Recommendations

The following recommendations are applicable to VSAM usage.

All VSAM Files

When a file is defined, VSAM considers it empty until a load operation (WRITE) is processed. Errors occur on all the VSAM supporting verbs (except WRITE) until this load operation has been processed.

The record length (VSE only) on the file specifications statements must contain the maximum record size defined in the VSAM catalog for the VSAM file being used.

The file I/O area contains high values (X'FF') when end-of-data is reached or on a record-not-found condition.

You must supply the record length on the WRITE, REWRITE, and ADDRECORD commands.

- You can examine the length of the record in VAL243-246-B after the record has been retrieved and before another VSAM supporting verb is executed.
- Another technique simply sets a pointer to the 4-byte length field prior to the statement referencing it.

```
    SET PTA INF1                    /*  Point PTA to first position of INF
    SET PTA DOWN 4                  /*  PTA now points to length field
030 MOVE P'80' TO PTA1-4-B          /*  Move in the length
```

Examine the length field:

```
030 IF PTAl-4-B EQ X'00000050'      /*  Examine it
```

If the record size is less than 32K, you may set the pointer down 2 and examine the 2-byte field.

- Rather than set PTx and then set it down every time, use the following technique once at the beginning of your VISION:Report program:

```
    OPEN INF                        /* Open the file for an I/O area
    SET PTA  INF1                   /* Point PTA to first position of INF
    SET PTA  DOWN   4               /* PTA now points to length field
040 SET PTA  SAVE                   /* Save PTA if it has other uses
          .
          .
          .
200 SET PTA RESTORE                 /* Restore PTA if it has other uses
210 MOVE P'80' TO PTAl-4-B          /* Move in the length
```

If PTx is dedicated to pointing at the length field only, statements 040 and 200 are not necessary. If you need to SAVE and RESTORE a pointer, either do not alter it (remember that there is only one save area per pointer) or save it to and restore it from an area in working storage.

KSDS Files

Record lengths may be changed using the REWRITE verb, but the key of the record cannot be changed, and the records must adhere to specifications as they are defined in the catalog.

A record may be deleted. The only requirement is that it must have been retrieved.

Note: For MVS only, the VSAM RBA returned can be 4 bytes for a non-extended format file or 8 bytes for an extended format file. See the VSAMER keyword in the OPTION section in Chapter 3. If VSAMER=YES, the VSAM error word is located 4 bytes to the right of the default location allowing the display of the 8-byte RBA.

ESDS Files

Any random access to one of these files requires a key as noted by the VSAM verbs. The key of an ESDS VSAM file is the RBA (relative byte address) of the record, and is always in 4-byte binary format. The RBA of a record just written or retrieved is available in VAL249-252-B and in VAL249-256-B for extended format. To view the 8-byte RBA, set Option VSAMER = YES to display the VSAM Error String in VAL257-259.

Records may only be added to these files with the WRITE verb, and are written in the next available entry position. The record sizes may not be changed once they have been added to the file, and the DELETE verb is not allowed.

RRDS and VRDS Files

Any random access to one of these files requires a key as noted by the VSAM verbs. The key of an RRDS VSAM file is the relative record number (or slot number) and is always in 4-byte binary format. The relative record number of a record just written or retrieved is available in VAL249-252-B.

When you are required to supply the record length or when you need to know the length of a record, one of the following techniques may be used to access the field that precedes the I/O area. You should exercise caution, because some of these techniques use VISION:Report index pointers.

- You can examine the length of the record in VAL243-246-B after the record has been retrieved and before another VSAM supporting verb is executed.
- Another technique simply sets a pointer to the length field prior to the statement referencing it.

```
SET PTA INF1                          /*  Point PTA to first position of INF

SET PTA DOWN 2                        /*  PTA now points to length field

030 MOVE P'80' TO PTA1-2-B            /*  Move in the length
```

Examine the length field:

```
    030 IF PTAl-2-B EQ X'0050'        /*  Examine it
```

- Rather than set PTx and then set it down every time, use the following technique once at the beginning of your VISION:Report program:

```
    OPEN INF                          /* Open the file for an I/O area
    SET PTA  INF1                     /* Point PTA to first position of INF
    SET PTA  DOWN   2                 /* PTA now points to length field
040 SET PTA  SAVE                     /* Save PTA if it has other uses
          .
          .
          .
200 SET PTA RESTORE                   /* Restore PTA if it has other uses
210 MOVE P'80' TO PTAl-2-B            /* Move in the length
```

Examine the length- field:

```
    210 IF PTAl-2-B EQ X'0050'        /* Examine it
```

If PTx is dedicated to pointing at the length field only, statements 040 and 200 are not necessary. If you are required to SAVE and RESTORE a pointer (remember that there is only one save area per pointer), do not alter it.

LDS Files

Any random access to one of these files requires a 4-byte word control interval number. The control interior number of record just written is not stored. Your program must retain this value.

VSE I/O File Parameter Statement
Positions 1-22 Are Fixed Form - Positions in 23-80 Are Free-Form

File Name 1-3	Media 4-7		Block Size 8-11	Record Size 12-15	LBL Info 16	Symbolic Unit 17-22	Options[1] 23-80
INF DET	CARD[2]		nnnn[3]	nnnn[3]		SYSnnn	General
INA-INB	TAPE	Fixed length tape					BF=N[4]
	TAPU	Undefined length tape					LBL=xxxxxxx[5]
INC	TAPV	Variable length tape			S=Standard		
	SD1C	2311 Disk fixed length			N=No Labels		BS=nnnnn[6]
IND	SD1V	2311 Disk variable length			blank=Default Standard Labels		RS=nnnnn[7]
INE	SD4C	2314 Disk fixed length					
	SD4V	2314 Disk variable length					
ING-INZ	SD3C	3330 Disk fixed length (Models 1 or 11)					Tape
OFA-OFZ	SD3V	3330 Disk variable length (Models 1 or 11)					TM=NO
	SD5C	3340 Disk fixed length					
	SD5V	3340 Disk variable length					REWIND=REWIND
	SD6C	3350 Disk fixed length					REWIND=NOREWIND
	SD6V	3350 Disk variable length					
	SD7C	3375 Disk fixed length					REWOMD=UNLOAD
	SD7V	3375 Disk variable length					
	DISC	3350, 3375, 3380, 3390, 9340/45 and up Disk fixed length					FILES=nnn[8]
	DISV	3350, 3375, 3380, 3390, 9340/45 and up Disk variable length					BP=nnnn[9]
	IS11	2311 Disk ISAM (INF, DET, OFA only)					Output ISAM OFA Only
	IS14	2314 Disk ISAM (INF, DET, OFA only)	Record size and block size specified must be the maximum that can occur for undefined variable files.				KLEN=nnn
	IS30	3330 Disk ISAM (INF, DET, OFA only)					KLOC= nnn
	IS40	3340 Disk ISAM (INF, DET, OFA only)					OFTRK=nn[10]
	FBAC	3330, 3370 FBA Disk fixed length					
	FBAV	3330, 3370 FBA Disk variable length					
	KSDS	KSDS VSAM File					
	ESDS	ESDS VSAM File					
	RRDS	RRDS VSAM File					
	VSAM		No parameters. Information is retrieved from catalog.				

1. Free-form format. A valid delimiter is a space or a comma. No comments are allowed in this field.

2. Data is read from SYSIPT, RL is 80. Positions 8-22 are ignored.

3. nnnn must be right aligned; leading zeros are optional.

4. BF=N causes no buffering for this file.

5. LBL=FILENAME causes STDLABEL track labels to be used.

6. nnnnn is a numeric value that is assigned to the BLOCKSIZE.

7. nnnnn is a numeric value that is assigned to the RECORD SIZE.

8. Multiple files may be read; maximum is 255.

9. (ASCII File) Block size must be 18-2048 bytes if BUFFER PREFIX LENGTH is specified.

10. Overflow tracks per cylinder; default is 2311-2, 2314-4, 3330-3, 3340-2.

VSE Parameter Statements

VSE I/O Parameter Statements - Examples

```
Position           Position
1                   22

INFTAPE36000080SSYS012
OFADISC40000200 SYS005
DETIS4010000100 SYS010
INFSD5Cl4000070 SYS016 LBL=SPECIAL
INFTAPE    1000SSYS011 BS=10000,REWIND=UNLOAD,LBL=OLDMAST,BF=N
INFTAPE        SSYS010 BS=5000,RS=100
INFTAPE20000200SSYS010 BP=26
DETTAPV40001200NSYS011 REWIND=NOREWIND,FILES=3,TM=NO
OFBSD5C02000200 SYS005
OFAIS1432000080SSYS010 KLEN=10,KLOC=l,OFTRK=2
INFCARD
OFAKSDS    0080
INJVSAM
```

Accessing More Than One Input File

VISION:Report has the ability to read more than one copy of like files through one of the input (tape only) files. When you want to read more than one file through one of the input files, you may code FILES=nnn any place after position 22 of the I/O parameter statement for the appropriate file (nnn is the number of files, not reels or volumes). If FILES=nnn is not used, a default of one file is assumed. The maximum number allowed is 255.

If the number of files varies from run to run, VISION:Report counts the number of tapes specified by the FILES= operand and quits after the number of input tapes has been exhausted. However, a large specification (such as, 100 or 200) can be entered for the number of tape input files and, after the last tape file has been read, the operator can reply EOJ causing VISION:Report to recognize this as the last tape.

VSE Input and Output Files

The VSE version of VISION:Report requires that all files being used for input or output be defined on the VSE I/O parameter statement. For 3350 and later devices (such as 3375, 3380, 3390, 9340/9345), use the generic media keywords DISC or DISV in columns 4-7 of the I/O parameter statement.

VSAM files can use either the VSAM file type KSDS, ESDS or RRDS with a record length, or VSAM without a record length, in which case, VISION:Report obtains the file type and record length from the VSAM catalog.

VISION:Report supports up to 53 files as follows:

Files	
DET	Detail Input File
INA-INZ	Input Files
OFA-OFZ	Output Files

VSE files are defined by a single semi-fixed format I/O parameter statement.

Any tape file you define to VISION:Report must be mounted unless the JCL for that file specifies ignore.

If the files are defined, the respective VISION:Report data areas DET, INA through INZ, and OFA through OFZ are available for use after the file is opened.

All file opens and closes are automatic, unless the OPEN and/or CLOSE statements are used.

The executable I/O statements that affect data transfer are:

Statements		
GET	WRITE	ADDRECORD
READ	REWRITE	DELETE

ISAM input files can be generically accessed or updated by the respective SETGENKEY and REWRITE.

User statement references to file areas are posted and checked after all statements have been read. A diagnostic occurs (ERR208) if a direct reference is made to an offset in a file that is greater than the record size specified in the I/O parameter statement.

VSAM files are supported natively. You can read sequentially or randomly, add to, change, or delete records from VSAM files.

VSE Block/Record Size

VSE block size and record size are taken from the I/O parameter statements. An example of a tape I/O statement follows:

```
INFTAPE32000080SSYS010
```

If the block size and/or record size is greater than four digits, leave the appropriate positions on the VSE I/O parameter statement blank and use the BS=nnnnn and/or RS=nnnnn options. An example follows:

```
INFDISC        SSYS020 BS=20000,RS=200
```

Block size and record size are restricted to the limitations of VSE.

VSE Fixed Length Files

Fixed length input files are processed directly in the input I/O area by setting an address pointer in the VISION:Report internal directly at the record to be processed. This mode of operation does not require a work area and eliminates data movement. This mode is often referred to as GET Locate.

The VSE block length specified can be larger than the actual block size on the file, as long as the record size is divided evenly into the actual block size.

VSE drops records with no warning if a block size is specified that is smaller than that of the file and is evenly divisible by the record.

Fixed length output files (except ISAM) are processed directly in the output I/O area by setting an address pointer in the VISION:Report internal directly at the next vacant record area in the I/O area.

You create the output record with one or more MOVE commands.

A WRITE OFx causes VISION:Report to write a block, if need be, and positions the pointer to the next vacant record.

A reference to the OFx areas, after a WRITE occurs, does not point to the data or record just written. It points to the next record area. (VSE only.)

VSE Variable Length Files

Variable length input files are processed with an I/O area, work area concept. The I/O area is acquired dynamically based on the block size specified; a work area is also acquired based on the record size specified.

The preceding explains why the block size and record size specified for variable files must be the largest block and record size to be found anywhere in the file.

The work area is optionally cleared to blanks prior to deblocking each record into the work area. This ensures that residue from previous records (that may be longer) is erased by VISION:Report. Variable length input must be of the form that contains the block length in the form of BLxx as bytes 1-4 of the block, and the record length as RLxx as bytes 1-4 of each record.

Variable length records are delivered to you in the work area with the record length (RLxx) in positions 1-4 of the work area. You must, of course, consider this in your specification of where a field starts. Initializing blanks to the work area is optional. (See the section OPTION, in Chapter 3 for installation defaults and the CLRVIP and CLRVOP keywords for details.)

Variable length output files use an I/O area and a work area concept. The VISION:Report internal pointer points to a work area acquired, the size of the largest variable record specified. You should build your output record consisting of RLxx in positions 1-4 and the data starting in position 5. RL must be in binary form. You should ensure that the binary record length you build into RL is accurate. When you issue the WRITE OFx statement, the output area is moved to the I/O area and the block is written when needed.

Variable length output areas may be blanked by VISION:Report before turning the area over to you for use; fixed length output areas are not. Preblanking is optional. See CLRVOP in the section OPTION, in Chapter 3.

VSE Undefined Files

No blocking is performed for input files. VISION:Report delivers the block size on input files at VAL1-4-P. Undefined records or blocks are read directly into an I/O area. An internal VISION:Report pointer points directly to the record in the I/O area with no data movement taking place.

The output is processed directly in the output I/O area.

No blocking is performed for output files. You should move the exact data record/block you want to OFA-OFZ. The length placed in VAL1-4-P prior to issuing the WRITE verb is the length to be used for creating the data on the output file.

VSE ISAM Files

Block size and record size are used to allocate I/O areas. ISAM input files are processed directly in the I/O area. In the case of unblocked ISAM files, the block size and record size should be specified as equal numbers; VISION:Report allocates sufficient extra bytes for the key and sequence link.

Actual reading and deblocking of the ISAM file is based on the block size and record size recorded in the file labels, stored in the VTOC of the ISAM file. Unblocked ISAM files must have the key embedded in the record to enable VISION:Report to deliver the key and data to you. If the key is not embedded in the record, the data-only portion of the record is all that is delivered to you. Block and record sizes are picked up from the I/O parameter statements.

ISAM output files are built using an I/O area and a work area. The work area is the size of your record length in the case of blocked files. The work area is the size of your record length plus your key length in the case of unblocked ISAM. Build your record into OFA using MOVE commands. The key should be placed in the output area consistent with the KLOC specification. If KLEN is specified as 10 and KLOC is specified as 1, you should build a 10-digit key in output positions 1-10.

Unblocked ISAM files require the key to be in position 1. If you specify an unblocked ISAM file of 80 position records with a 15 position key, the work area is 95 positions; build your key in 1-15 and your data in 16-95.

ISAM Output Parameters for OFA

To load an ISAM file, you must specify the following options on the OFA output parameter statement:

```
KLEN=nnn,KLOC=nnn,OFTRK=nnn
```

The KLEN parameter specifies the length of the key for the ISAM file.

The KLOC parameter specifies the location of the key within the record. Unblocked ISAM files do not have the key contained within the record. The KLOC parameter for unblocked ISAM files should be KLOC=001.

For blocked ISAM files, the key may be contained within the record and does not have to be in position 1. If your key starts in position 12 of a blocked ISAM file, the KLOC parameter would be KLOC=012.

The OFTRK parameter specifies how many overflow tracks to reserve on each cylinder. The OFTRK parameter should never be specified to be equal to or greater than the number of tracks in a cylinder; it should always be less. If OFTRK is not specified, the default is as follows:

DEVICE	OVERFLOW TRACKS
2311	2
2314	4
3330	3
3340	2

OFTRK is specific to ISAM. ISAM is not supported on 3350 and later devices.

VSE Considerations

FILE statements are required for input/output files (no PRT or PUN definitions are needed). SYSIPT, SYSLST, and SYSPCH are used for reading in VISION:Report program statements or data, printer output, and punch output, respectively.

Files accessed by using the CALL routines (QUIKISAM, QUIKVSAM, QUIKDLI, QUIKIDMS) do not require I/O statements, although the appropriate JCL statements are needed.

VISION:Report Parameter Statements

Input/output parameter statements (DET, INA through INZ, and OFA through OFZ) are required. See VSE Parameter Statements.

Execute Statement

```
// EXEC QUKBJOB
```

MVS Input and Output Files

VISION:Report supports up to 53 files as follows:

Files	
DET	Detail Input File
INA-INZ	Input Files
OFA-OFZ	Output Files

MVS files are defined through the DD JCL as follows:

DD JCL	
DET	SYSDET
INA	SYSINA
INB	SYSINB
INC	SYSINC
IND	SYSIND
INE	SYSINE
INF	SYSUT1
ING-INZ	SYSING-SYSINZ, respectively
OFA	SYSUT2
OFB	SYSUT3
OFC	SYSUT4
OFD	SYSUT5
OFE-OFZ	SYSOFE-SYSOFZ, respectively
PRT	SYSPRINT
PUN	SYSPUNCH

If the respective files are defined, the VISION:Report data areas DET, INA-INZ, and OFA-OFZ are available for use. (To change the defaults, see the section OPTION, in Chapter 3.)

All files are opened and closed automatically, unless the OPEN and/or CLOSE statements are used.

The imperative I/O statements that affect the actual data transfer are:

I/O Statements		
GET	WRITE	ADDRECORD
READ	REWRITE	DELETE

Input files can be limited by the LIMITREADS and SAMPLES declarative statements.

ISAM input files can be generically accessed or updated by the respective SETGENKEY and REWRITE.

User statement references to file areas are posted and checked after all statements have been read. A diagnostic occurs (ERR208) if a direct reference is made to a file that is greater than the LRECL value taken from the DD statement or data set label.

VSAM files are supported natively. You can read sequentially, read randomly, add, change, or delete VSAM records.

MVS Block/Record Size

MVS block size and record size are taken from either the DD statement (if present) and/or from the data set labels. A DD statement example to read a tape input file follows:

```
//SYSUT1 DD DSN=file.id,DISP=OLD,VOL=SER=volnum,UNIT=TAPE
```

MVS Fixed Length Files

Fixed length input files are processed directly in the input I/O area by setting an address pointer in the VISION:Report internals directly to the record to be processed. This mode of operation does not require a work area and eliminates data movement. This mode is often referred to as GET Locate.

All fixed length output files are processed in a work area that is the same size as the LRECL of the file.

You assemble your output record with one or more MOVE statements.

A WRITE OFx causes VISION:Report to transfer the work area to the output buffer. This mode is referred to as PUT MOVE.

After a WRITE, the record just written is still available in the work area. The following code is valid (under MVS):

```
010     GET
        MOVE INF1-80 TO OFA1
        WRITE OFA                       /* Write first record
        MOVE C'2' TO OFA1               /* Modify record
        WRITE OFA                       /* Write second record
        GOTO 010
```

MVS Variable Length Files

Variable length records are always delivered to you with the record length (RLxx) in positions 1-4. You must consider this in determining where the actual data starts.

Variable input files are processed using one of two methods:

- Directly in the input I/O area. This is accomplished by setting an address pointer, in the VISION:Report internals to the record that is to be processed. This method of processing is not used if the file is VBS (variable blocked spanned). It is used only if the OPTION CLRVIP=NO (default) is in effect.
- Using a work area that is the same size as the LRECL of the file if the file is VBS and/or the OPTION CLRVIP=YES is in effect. The internal VISION:Report pointer always points to this area.

Variable length output files use an I/O area and a work area concept. The VISION:Report internal pointer points to a work area acquired, the size of the largest variable record specified. You should build your output record consisting of RLxx in positions 1-4 and the data starting in position 5. RL must be in binary form. You should ensure that the binary record length you build into RL is accurate. When you issue the WRITE OFx statement, the output area is moved to the I/O area and the block is written when needed.

Variable length output areas may be blanked by VISION:Report before turning the area over to you for use; fixed length output areas are not. Initializing blanks to the work area is optional. See CLRVOP in the section OPTION, in Chapter 3.

MVS Undefined Files

Undefined input and output files may be tape or disk for MVS.

No blocking is performed for input files. VISION:Report delivers the block size on input at VAL1-4-P. Undefined records or blocks are read directly into an I/O area. An internal VISION:Report pointer points directly to the record in the I/O area with no data movement taking place.

No blocking is performed for output files. You should move the exact data record/block you want to OFA-OFZ. The length placed in VAL1-4-P prior to issuing the WRITE verb is the length to be used for creating the data on the output.

MVS ISAM Files

Block size and record size are used to allocate I/O areas. ISAM input files are processed directly in the I/O area. In the case of unblocked ISAM files, block size and record size should be specified as equal numbers; VISION:Report allocates sufficient extra bytes for the key and sequence link.

Actual reading and deblocking of the ISAM file is based on the block size and record size recorded in the file labels, stored in the VTOC of the ISAM file. Unblocked ISAM files must have the key embedded in the record to enable VISION:Report to deliver the key and data to you. If the key is not embedded in the record, the data-only portion of the record is all that is delivered to you. Block and record sizes are picked up from the DD statement and/or the file itself.

For ISAM files, RECFM=F and RKP=0 are not allowed.

ISAM output files are built using an I/O area and a work area. The work area size is equal to your record size. Build your record into OFA using MOVE commands. The key should be placed in the output area consistent with the key location of the file. If key length of the file is specified as 10 and the key location is specified as 1, you should build a l0-digit key in output positions 1-10.

MVS Considerations

VISION:Report Parameter Statements

Input/Output parameter statements (DET, INA-INZ, and OFA-OFZ) are not required.

Functional Differences

No VISION:Report MVS messages are issued to the console. All messages are routed to the printer (SYSPRINT) output.

If VISION:Report MVS ABENDs with code U3333, see the printer (SYSPRINT) output data set for an explanation.

Execute Statement

```
//stepname EXEC PGM=QUIKJOB
```

DD Statement and File Characteristics

The following table details the relationship between VISION:Report MVS file names and ddnames, and DCB characteristics of each data set.

Report File	DDNAME	RECFM	LRECL	BLKSIZE	QISAM DCB INFO,RKP, KEYLEN, and so on
DET	SYSDET	Note 1	Note 1	Note 1	Note 1
INA-INE	SYSINA-SYSINE	Note 1	Note 1	Note 1	Note 1
INF	SYSUT1	Note 1	Note 1	Note 1	Note 1
ING-INZ	SYSING-SYSINZ	Note 1	Note 1	Note 1	Note 1
OFA	SYSUT2	Note 2	Note 2	Note 2	Note 3
OFB	SYSUT3	Note 2	Note 2	Note 2	Note 3
OFC	SYSUT4	Note 2	Note 2	Note 2	Note 3
OFD	SYSUT5	Note 2	Note 2	Note 2	Note 3
OFE-OFZ	SYSOFE-SYSOFZ	Note 2	Note 2	Note 2	Note 3
PRT	SYSPRINT	FBA	Note 5	Note 4	N/A
PUN	SYSPUNCH	FBA	Note 6	Note 4	N/A
	SYSIN	FB	80	Note 1	N/A

Table Notes

1. For non-VSAM data sets, taken from the DD statement or input data set label. For VSAM data sets, taken from VSAM catalog.
2. Copied from the DD statement for this file or from the SYSUT1 file (if present, not VSAM, and file is opened prior to output file — this method is not recommended, because of the enormous potential for errors).
3. Must be specified in the DD statement for this file.
4. Same as LRECL unless specified in the DD statement for this file.

5. This is the user option with a default of 133.
6. This is the user option with a default of 81.

SYSIN and SYSPRINT DD statements are required for all executions. SYSPRINT can be overridden by the OPTION PRTDD verb. Others are required only if referenced in parameter statements by the GET/WRITE/PUNCH verbs or use of the file area.

All files may reside on any tape, disk, or unit record device supported by QSAM and may be any RECFM. In addition, SYSUTx files may be QISAM. You must include DCB=(DSORG=IS), along with any other necessary QISAM DCB subparameters on the appropriate DD statements, if the data set is QISAM.

Concatenation of data sets with unlike attributes is allowed. LRECL and format (fixed, variable, undefined) must be the same.

All other DD keyword requirements are MVS standard.

Database Files

For the databases not supported directly by VISION:Report, such as QUIKDLI, QUIKIDMS, and QUIKIMS, you may build a calling parameter for the appropriate routine, as described in the DLI, IDMS, or IMS reference manual. For DB2 or SQL/DS rows, use the VISION:Report EXEC SQL verb, as described in the VISION:Report Interface to DB2® Reference Guide.

QUIKDLI, QUIKIDMS, QUIKIMS, and VISION:Interface for DB2 and SQL/DS with VISION:Report are optional interfaces and may not be available at your installation.

DB2 or SQL/DS

DB2 or SQL/DS rows or files may be accessed by using the VISION:Report EXEC SQL verb. See the VISION:Report Interface to DB2 Reference Manual.

ADABAS Interface

ADABAS access is accomplished by an access module, named ADAMINT. From your VISION:Report program, build a calling parameter list to ADAMINT as described in the appropriate ADABAS reference manual.

TOTAL Interface

TOTAL®, by its nature is callable. Therefore, TOTAL can be called directly from VISION:Report by using the CALL verb. For additional information, refer to your TOTAL reference manual.

DBOMP

See the section Optional Material, in Chapter 3 for calling sequences of DBOMP. (VSE only.)

User Abend Codes

For a complete list of diagnostic codes and user ABEND codes, refer to the VISION:Report Messages and Codes Guide.

Setting the Step Return Code

Move an EBCDIC value in the range 0000 through 4095 to VAL46-49 before EOJ. If you do not alter VAL46-49 during your execution, the return code is 0000 or whatever value was returned in register 15 from the last called program.

Causing the Step to Abend

Execution of a VISION:Report ABEND statement causes VISION:Report/MVS to issue an ABEND macro with the user ABEND code that you have placed at VAL67-70. The user ABEND code must be an EBCDIC value in the range of 0000 through 4095. If you do not alter VAL67-70 during your execution, it contains 4095. Do not use a code that begins with 3 as VISION:Report uses these codes.

A hexprint of the active VISION:Report areas can be indicated by setting the option QJMDUMP=YES when an ABEND is issued.

A memory dump is produced when option UABNDMP=YES and an ABEND is issued.

VISION:Report can provide memory dumps for each of the above CANCEL code conditions. This is done with the YES/NO settings of the VISION:Report Options U331DMP-U399DMP. (See the section OPTION, in Chapter 3.)

Chapter

3

Statement Format

General Rules

Use the following rules for the syntax of VISION:Report. In most cases, you must enter statements in the exact order given.

Rule	Description
Rule A	Brackets [ ] indicate an optional entry.
Rule B	Braces { } indicate a choice of entries; unless a default is indicated, you must choose one of the entries.
Rule C	Items separated by a vertical bar | represent alternative items. You can select no more than one of the items.
Rule E	An ellipsis ... indicates that multiple entries of the type immediately preceding the ellipsis are allowed.
Rule F	Underscored type indicates the default entry. If the operand is omitted, the underscored value is assumed.
Rule G	Uppercase letters indicate the characters to be entered. You must enter such items exactly as shown.
Rule H	Lowercase letters indicate fields you must supply.
Rule I	You must enter punctuation, such as parentheses and single quotation marks, exactly as shown.
Rule J	A sequence number (seq-no) can preface each statement. The sequence number can be 1-8 digits, except in the HDR statement, where it can be 1-4 digits.
Rule K	A period indicates the end of true condition processing (IF or WHEN) and identifies the next statement as the beginning of false processing (ELSE).

Rule	Description
Rule L	The following variations are acceptable: ZERO, ZEROS, or ZEROES; BLANK or BLANKS; SPACE or SPACES; HIVALUE or HIVALUES; LOVALUE or LOVALUES; SINGLE SPACE, SINGLE SPACED, SINGLESPACE, or SINGLESPACED; DOUBLE SPACE, DOUBLE SPACED, DOUBLESPACE, or DOUBLESPACED; TRIPLE SPACE, TRIPLE SPACED, TRIPLESPACE, or TRIPLESPACED; ZERO SPACE, ZERO SPACED, ZEROSPACE, or ZEROSPACED; EQ, EQUAL, or =; GT or >; LT or <.

ABEND

```
ABEND [code]
```

Term	Description
ABEND	Cancels VISION:Report with a user-chosen ABEND code. The ABEND statement causes VISION:Report to issue an MVS ABEND macro or a VSE CANCEL macro to cancel the job and bypass the remaining steps in the same job.
code	Either move an ABEND code to VAL67-70 prior to issuing the ABEND statement or specify a code here. The ABEND code must be between 0000 and 4095. Codes beginning with 3 are not recommended, as VISION:Report uses these for other purposes. VSE VISION:Report prints this code or the ABEND code from VAL67-70 on the printer prior to issuing the CANCEL macro.

If no ABEND code is specified or VAL67-70 is not altered, the code used is 4095. A code moved to VAL67-70 that is greater than 4095 results in 4095.

ACCEPT

```
ACCEPT flddef
```

Term	Description
ACCEPT	Transfers to the defined field a data stream keyed into the system through the operator's console.
flddef	Define the field to receive the data keyed into the system through the console. This field can be a maximum of 80 positions.

The ACCEPT statement causes suspension of the application until the operator has keyed a data stream followed by an end of block command or the appropriate terminator (such as the ENTER key).

The operator's reply is forced to uppercase and placed in the user-defined area, and the application resumes execution.

Example

In this example, if the operator keys 10 characters before the EOB, 10 characters are entered starting in WST1. Prior contents of all 10 positions are overwritten.

```
010 DISPLAY C'ENTER MONTH OF PROCESS AS NN'       /* Notify operator
020 ACCEPT WST1-2                                /* Accept operator reply
030 IF WST1-2 IS NUMERIC                         /* Test data for numeric
040      GO TO ...                               /* or go to the appropriate
050                                              /* processing routine
```

ACCUM

The ACCUM statement can be used in two contexts: accumulation and automatic printing of totals (called a simple summary) and a more versatile context (called summarize and calculate), which allows you to reference each counter directly and perform calculations at control breaks.

Simple Accumulation

If no references to accumulators for evaluation and/or calculation are made at total time, the general accumulator CTR is used in all ACCUM statements.

This action causes VISION:Report to automatically assign the first counter to the first ACCUM statement, and the second counter to the second ACCUM, as examples.

There are 16 CTR accumulators. Each contains an 8-byte, packed decimal value.

No direct references to these accumulators are allowed. You cannot accumulate two or more data fields into the same accumulator without first forming the value into the field specified in the ACCUM statement.

Addressable Accumulation

When either detail-time reference to accumulators (such as, ADD, SUB, IF) or control break calculations-before-total-print is intended, you specify one of 16 accumulators CTA, CTB, CTC, . . . CTP for each ACCUM. This method provides direct addressability to the minor level counters at detail time. Using this form of the ACCUM, you accumulate from more than one field into one accumulator using ACCUM, ADD, SUB, MULT, and DIVD statements.

If control break calculations are involved (invoked through CHECKBREAKS ON BREAKS PERFORM xxx THRU yyy) you may place calculation results into accumulators that have not been accumulated. In this case, you code an ACCUM NONE INTO A n BYTE CTx statement. This dummy ACCUM provides total print specifications for CTx, but generates no detail-time accumulation coding.

The two different contexts cannot be coded in the same program. If CTR is specified in one ACCUM, you must specify it in all ACCUM statements.

ACCUM (with REPORT)

```
ACCUM {flddef|ONE|NONE} [IN A n BYTE CTx]
```

Term	Description
ACCUM	Accumulates and prints totals automatically in a report. The totals are lined-up in the proper detail columns. This eliminates the need to specify print positions in the ACCUM statement when using it with the REPORT statement.
flddef	Specifies the data field that is to be accumulated.
ONE	Accumulates a count of one each time the program executes the ACCUM ONE statement. The length of the counter and the CTx must be specified for access to the counter. To print this total, the CTx (or its equated name) must be referenced in the REPORT statement, with the number of bytes specified.
NONE	Performs no accumulations. Specifying NONE reserves an accumulator for later data storage and/or use. To print the contents of this CTx, it must be referenced in the REPORT statement and IN A n BYTE CTx must be specified. A message, **Period Ignored** is issued if a period follows the statement ACCUM NONE. The ACCUM NONE is declarative and therefore cannot be the subject of a transfer in control statement (for example, GOTO, PERFORM).
IN A n BYTE CTx	IN A n BYTE provides the number of bytes for the length of the counter (from 1 to 8). CTx defines the accumulators CTA through CTP.

The two forms of ACCUM statements (with or without IN A n BYTE CTx) cannot be combined within the same program. Apply the chosen form consistently in each ACCUM statement.

If IN A n BYTE CTx is specified, you must also specify CTx with the correct number of bytes and the rightmost position on the REPORT statement.

For example:

```
REPORT CTA7-8-P 0
     .
     .
ACCUM ONE IN A 2 BYTE CTA
```

If access to the accumulated total is not required, and you want to accumulate and print only, IN A n BYTE CTx is not necessary.

The number of print positions generated for each accumulator is calculated as the length of your source field plus 2 digits. If the total requires more print positions than the source field plus 2, the IN A n BYTE CTX operand gives you override control of the number of print positions.

Example

The input field is PENNIES (2 bytes). But, if the expected total exceeds one million dollars, you would code:

```
ACCUM PENNIES IN A 6 BYTE CTA.
```

The ACCUM, REPORT, BREAK, and CHECKBREAKS statements work together to produce a report with totals.

The data names to be accumulated must first be referenced by the REPORT statement.

```
EQU PLANT INF2-4
    EQU EMP-NUM INF7-10
    EQU EMP-NAME INF20-39
    EQU SSN INF11-19 S
    EQU HR-RATE INF50-54 2C
    EQU YR-RATE INF55-59 0C
    TITLE1 'COMPUTER ASSOCIATES'
    TITLE2 'PERSONNEL REPORT'
    REPORT PLANT (PLANT-NAME) EMP-NUM (EMPLOYEE-NUMBER)
      EMP-NAME (EMPLOYEE-NAME)
      SSN (SOCIAL-SECURITY-NUMBER)
      HR-RATE (HOURLY-RATE)
    BREAK 1 PLANT SB 1 SA 1

010 GET INF
    CHECKBREAKS
    ACCUM HR-RATE
    PRINT REPORT
    GOTO 010
    END
```

```
                              COMPUTER ASSOCIATES
                              PERSONNEL REPORT

PLANT    EMPLOYEE    EMPLOYEE            SOCIAL        HOURLY
NAME       NUMBER      NAME            SECURITY        RATE
                                         NUMBER

CHI          6211   KELLAR,CORETA      678-72-5411      .00
CHI          4396   CLINGMAN,GREGORY   949-09-1052    11.00
CHI          1999   MANNING,PAULA      861-22-2579      .00
CHI          1918   PIRNIA,LINDA       226-18-5824    10.60
.          .        .                      .              .
.          .        .                      .              .
.          .        .                      .              .
                                                      21.60

SFR          2460   IVANOFF,SUSANA     241-98-8324    10.00
SFR          4546   RAHL,THOMAS        172-76-1610    10.60
SFR          0632   TITAN,MARKSIX      948-10-1300      .00
SFR          1711   SHERIDAN,HONEY SUE 676-10-8110    11.25
SFR          1219   JOHNSON,HAROLD     503-28-2966      .00
SFR          0911   THOMPSON,PETER     172-13-1972      .00

                                                      31.85

                                                      53.45
```

ACCUM (User Addressable)

```
ACCUM {flddef|ONE|NONE} [IN A n BYTE CTx [ON BREAKS PRINT IN POS nnn [nC|nE|nN]]]
```

Term	Description
ACCUM	Adds data, a value of one, or none (defines printing specifications only, does not accumulate) to one of 16 VISION:Report-generated accumulators. This form of the ACCUM statement allows you to address accumulators. These are printed at a control break by VISION:Report. The ACCUM statement is coded at the point where the actual add to the accumulator is performed.
flddef ONE NONE	Enter the flddef of the field to be accumulated, or enter ONE or NONE. The data field can be in EBCDIC format (maximum of 19 digits), packed format (maximum of 10 bytes), or binary format (maximum of 8 bytes). A data field can be EBCDIC and have leading or all blanks. VISION:Report adds leading zeros before accumulating. Data that is not blank or numeric causes the program to ABEND. You should include IF...NUMERIC coding ahead of the ACCUM to ensure that only valid data reaches the ACCUM. If you enter ONE, a count of one is accumulated each time the program executes the ACCUM statement. This form is useful for counting purposes. If you enter NONE, there is no accumulation performed. VISION:Report sets up print specifications for the accumulator, assuming you will calculate and add a value to the accumulator at the control break. VISION:Report automatically performs the functions such as prints the accumulator, rolls it and clears it. A message, **Period Ignored** is issued if a period follows the statement ACCUM NONE. The ACCUM NONE is declarative and therefore cannot be the subject of a transfer in control statement (such as, GOTO, PERFORM).
IN A n BYTE Ctx	IN A n BYTE provides the number of bytes for the length of the counter (from 1 to 8). This length is necessary in order that the appropriate edit word is constructed for printing totals. There are 16 accumulators (and 16 ACCUM statements) available. CTx defines the accumulators CTA through CTP.

Term	Description
ON BREAKS PRINT IN POS nnn	Leftmost print position in the total lines where the accumulated total appears. The printed total is leading zero-suppressed and contains a decimal point to the left of the position specified in [nC, nE, nN]. If you are detail printing the values being accumulated and intend to print the totals aligned with the detail values, you should carefully align the detail and total print positions.
nC, nE, nN	n defines the number of decimal positions to be shown in the printed total. Enter 0 through 9. C, E, and N are optional. C inserts commas as appropriate. E (European) inserts a period before every third integer position, and a comma in the decimal position. N (non-zero suppression) suppresses leading zeros. The area required in the PRT area is: ■ One byte for each data byte. ■ One byte for a minus sign if CRSIGN=NO. Two bytes for the literal CR if CRSIGN=YES. ■ One byte for a decimal point if decimals are specified. ■ One byte for every three integers if C or E is specified.

The BLANK WHEN ZERO (BWZ) option suppresses printing zero-filled counters.

- If OPTION BWZ=NO is in effect, an ACCUM statement specifying 2C prints as .00.
- If OPTION BWZ=YES is in effect, an ACCUM statement specifying 2C prints as a blank field.

You also have the option of designating whether or not negative amounts in counters print with a negative sign (-) or credit character (CR).

- If OPTION CRSIGN=NO causes a negative sign to print, indicating a negative counter.
- If OPTION CRSIGN=YES causes CR to print, indicating a negative counter count. The use of CR also requires one additional position to be used in the print line.

See MOVE for printing space and edit mask requirements.

ACCUM (Simple Accumulation)

```
ACCUM {flddef|ONE} [IN A n BYTE CTx [ON BREAKS PRINT IN POS nnn [nC|nE|nN]]]
```

Term	Description
ACCUM	Adds data or a value of one to VISION:Report-generated non-addressable accumulators which are printed automatically at the control break. This form of the ACCUM statement is non-addressable, simple accumulation. Code the ACCUM statement at the point where the addition to the accumulator is to be performed.
flddef ONE	Enter the flddef of the field to be accumulated, or enter ONE. The data field is either EBCDIC, packed, or binary. Binary fields must be 1-8 bytes long. If the data field is EBCDIC and has leading or all blanks, VISION:Report adds leading zeros before accumulating. Data that is not blank or numeric causes the program to ABEND. Include IF...NUMERIC coding ahead of the ACCUM to ensure that only valid data reaches the ACCUM. If you enter ONE, a count of 1 is accumulated each time the program passes through the ACCUM statement.
IN A n BYTE Ctx	IN A n BYTE provides the number of bytes for the length of the counter (from 1 to 8). This length is necessary to construct the appropriate edit word for printing totals. There are 16 accumulators (and 16 ACCUM statements) available. CTx defines the accumulators CTA through CTP.
ON BREAKS PRINT IN POS nnn	Leftmost print position in the total lines where the accumulated total appears. The printed total is leading zero-suppressed and contains a decimal point to the left of the position specified in [nC, nE, nN]. If you are detail printing the values being accumulated, and intend to print the totals aligned with the detail values, align the detail and total print positions.

Term	Description
nC, nE, nN	n defines the number of decimal positions to be shown in the printed total. Enter 0 through 9. C, E, and N are optional. C inserts commas as appropriate. E inserts a period before every third integer position and a comma in the decimal position. N suppresses leading zeros. The area required in the PRT area is: ■ One byte for each data byte. ■ One byte for a minus sign if CRSIGN=NO. Two bytes for the literal CR if CRSIGN=YES. ■ One byte for a decimal point if decimals are specified. ■ One byte for every three integers if C or E is specified.

The BLANK WHEN ZERO (BWZ) option suppresses printing zero-filled counters.

- If OPTION BWZ=NO is in effect, an ACCUM statement specifying 2C prints as .00.
- If OPTION BWZ=YES is in effect, an ACCUM statement specifying 2C prints as a blank field.

You also have the option of designating whether or not negative amounts in counters print with a negative sign (-) or credit character (CR).

- If OPTION CRSIGN=NO causes a negative sign to print, indicating a negative counter.
- If OPTION CRSIGN=YES causes CR to print, indicating a negative counter count. The use of CR also requires one additional position to be used in the print line.

See MOVE for printing space and edit mask requirements.

```
OPTION SEQCHK=NO
EQU  GROSS INF3-5-P
EQU  HOURS-WORKED  INF6-9
EQU  P-PLANT  PRT1
EQU  PLANT  INF1-2
Break 1 PLANT SB N……
100 GET
    CHECKBREAKS
    MOVE PLANT TO P-PLANT
    ACCUM HOURS-WORKED IN A 4 BYTE CTR, ON BREAKS PRINT IN POS 004 2C
    GO TO 100
```

This produces a report of summary total lines at the plant level. You supply HDR statements for the headings.

```
PLANT        HOURS-WORKED            GROSS
  26           X-X.XX                X-X.XX
```

ADD

```
ADD {flddef1|C'xxx'|P'nnn'|X'...'} TO flddef2
```

Term	Description
ADD	Adds the first value to the second value.
flddef1 C'xxx' P'nnn' X'...'	Defines the flddef or literal with a limit of 19 digits EBCDIC or a packed field of 10 bytes (19 digits and sign). Binary fields can be 1-8 bytes. Fields and literals in any data format can be added to fields in any other data format. Automatic data conversion is performed in all cases.
TO	Required noise word.
flddef2	Defines the target area/field with a limit of 19 digits EBCDIC or a packed field of 10 bytes (19 digits and sign). Binary fields can be 1-8 bytes.

Examples

```
ADD INF1-19 to WST1-19                  /* Add EBCDIC to EBCDIC
ADD INF1-10-P TO OFA10-13-P             /* Add packed to packed
ADD WST1-3-B TO WST11-16-B              /* Add 3-byte to 6-byte binary
ADD INF8-10-P TO PUN1-10                /* Add packed to EBCDIC
ADD INF47-50-B TO OFA2l-26-P            /* Add 4-byte binary to packed
ADD C'1' TO WST5-7                      /* Add a literal to EBCDIC
ADD C'123' TO OFA7-10-P                 /* Add a literal to packed
ADD C'4' TO OFF1-8-B                    /* Add a literal to 8-byte binary
```

ADD treats fields with leading or all blanks (EBCDIC format) as zeros in the blank positions.

ADDRECORD

```
ADDRECORD {INA-INZ|DET} [flddef] [ONERROR seq-no]
```

Note: This statement cannot be used with ESDS VSAM files.

Term	Description
ADDRECORD	Inserts a record into a KSDS or RRDS VSAM file based upon the key of the record, if KSDS, or the relative record number, if RRDS.
INA-INZ, DET	File name can be any VISION:Report input file name (INA-INZ, DET).
flddef	Optional. A field definition or an equated data name that contains the relative record number (key) of the record. This is valid for RRDS VSAM files only and must be coded when the file is RRDS. The key field must be 4-byte binary. The key for a KSDS VSAM file must be within the record.
ONERROR seq-no	Optional. The ONERROR operand causes VISION:Report to automatically transfer to the statement of the specified sequence number when a VSAM error occurs. This allows you to examine the error codes (RC/EC) and take appropriate action. If this operand is not coded and a VSAM error occurs, execution continues with the next statement. It then becomes your responsibility to check for these errors.

You must specify the length of the record in the 2-byte field preceding the I/O area. See the techniques in the section File Specifications and Data Definitions, in Chapter 2.

The following error is returned when the ONERROR operand is not coded:

```
                    Error Word          RC/EC
Error               VAL253-255          VAL 247-248

DUPLICATE RECORD    'DUP'               X'0808'
```

AND (Logical And)

```
AND flddef1 WITH {flddef2 | C'xxx' | P'nnn' | X'...' | ZERO | BLANK | SPACE |
HIVALUE | LOVALUE } { flddef3 | X'...' }
```

Note: Do not confuse this AND verb with the AND option on the IF verb.

Term	Description
AND	Performs the logical AND of a data field. The contents of a bit position in the source are set to 1 if the corresponding bit positions in both operands contain ones; otherwise the bit is set to 0. The resulting condition code is returned in VAL223-B. If any bits in the source operand are 1 following completion of the operation, then the condition code is x'01'; otherwise, it is x'00'. In the case of a ZERO figcon, if flddef1 is character, then a character zero x'F0' is used; otherwise, a binary zero x'00' is used.
flddef1	Source data. This field can be changed because of the operation.
WITH	Required noise word.
flddef2 C'xxx' P'nnn' X'…' ZERO BLANK SPACE HIVALUE LOVALUE	The second operand. This field is not changed.
flddef3 X'…'	Optional. A 2- or a 4-byte binary field indicating the number of bytes to be ANDed. If this operand is omitted, then the length of the source field is used.

Term	Description

Example:

```
EQU FILLER       WST0
EQU FLDA         (6)   C'ABCDEF'
EQU FLDB         (6)   X'BFBFBFBFBFBF'
    AND FLDA WITH FLDB
    PRINTHEX FLDA
    PRINTHEX VAL223

                WST1-6
                       888888
                       123456
                      01..05.
            VAL223-223
                       0
                       1
                      01
```

ATEND

```
ATEND seq-no
```

Term	Description
ATEND	Allows you to control when the default input file INF reaches EOF (end of file).
seq-no	The statement number of the first statement of the end-of-job routine. The specified routine must end with GOTO EOJ. The statement number is a 1- to 8-digit sequence number followed by one or more spaces.

Use the ATEND statement to perform an end-of-input routine.

The ATEND is not normally needed in data selection programs or printed summary reports, as VISION:Report forces end of job processing. However, if you are summarizing through your own coding, you will lose your final summary as user coding is not ordinarily entered when end of input is recognized.

ATEND is not functional when any input file other than the default input file INF is being read. When any input file other than the default input file INF is specified, it becomes your responsibility to recognize EOF for all input files either by using the ATEND operand on the GET statement or by determining when any or all inputs have reached EOF (check for high-values in input area or test for E in VAL-IN?-EOF area).

The ATEND EOJ statement is not automatic with DET, INA-INE, and ING-INZ.

The ATEND statement is mutually exclusive with the ATEND operand on the GET statement. An error occurs if both are found in the same VISION:Report program.

If the ATEND statement is found when using two or more inputs and the ATEND operand was not used on any of the GET statements, the ATEND statement does nothing and is not diagnosed as being an error.

Automatic Summary Reporting

A significant number of report runs involve reading a sequenced file with group control fields, accumulating totals based on detail data and control breaks, and printing the group totals.

This type of program usually has certain standard logic: accumulation of totals as the detail records are read, testing for control breaks and printing totals, adding them to the next higher level, and zeroing the accumulators.

VISION:Report automates this process with three statements:

Term	Description
BREAK	A declarative statement that identifies the control (break) field positions in the input records. See BREAK.
CHECKBREAKS	An imperative statement that causes VISION:Report to compare the control fields in the current record (as described in BREAK statements) against the same fields in the last INF record. When control breaks are found, VISION:Report prints the accumulated totals, adds these to the next level accumulators, and zeros the accumulators. This print, add, zero cycle is repeated through the highest break level found. See CHECKBREAKS.
ACCUM	Both a declarative and imperative statement that causes data fields to be accumulated and provides VISION:Report with the total time print specifications for the accumulator. See ACCUM.

BREAK

```
BREAK {n|F} area [SB {n|E}] [SA {n|E}] [PRINT C'character string' [IN TOT POS nnn]]
```

Term	Description
BREAK	Defines a VISION:Report field that is to be compared to the contents of the same field in the last accepted record (which has been automatically saved by VISION:Report). A total break is implied whenever the contents of this field changes. Up to 9 total break levels, plus final totals, are allowed.
n | F	BREAK 1 is the minor break. BREAK 9 is the major break. Any number of fields may be examined for a break at each level. A break in any of the fields causes a break for the level specified. BREAK F allows a character string to be put in the final totals line; the SB and SA may be included, but the values are ignored.
area	You must always specify a VISION:Report area, normally an input file field, as break control.
SB n | E	SB stands for Space Before total print. Enter 0, 1, or 2 to generate zero, one, or two blank lines between the last detail line and the total line. Enter E to eject to the next page and print headers before printing totals. VISION:Report automatically blanks the PRT area after every user PRINT statement.
SA n | E	SA stands for Space After total print. Enter 0, 1, or 2 to generate zero, one, or two blank lines after the total line. Enter E to cause a page eject after printing this total line.
PRINT C'xxx'	The character string is an EBCDIC constant (up to 50 positions) that prints in the total line starting in the position specified in the total position operand (IN TOT POS nnn). This is ordinarily used to identify total levels (for example, PLANT TOTAL).
IN TOT POS	This operand defines the starting print position (nnn) for the constant defined in the PRINT C operand.

When the BREAK statement is used along with the REPORT statement, this operand is overridden, so that you do not have to assign a print position and do not overlay the totals. The print position is assigned by the REPORT statement by searching from left to right on the print line until space is found for the literal.

If you want to print summary or total lines only and do not want to print detail lines, use the SB version.

Move the identifying data to the PRT area such as Plant, Dept; do not issue a PRINT statement. The identifying data is left in the print line along with accumulated totals.

CHECKBREAKS

BREAK statements define the fields to break on the actual comparison; breaking is not executed until a CHECKBREAKS statement is encountered.

Example

```
010 BREAK 1 INF...              /* This sequence considers all
    020 GET                     /* Input records for break compares
    030 CHECKBREAKS



    010 BREAK 1 INF...          /* This sequence bypasses all
    020 GET                     /* Records that do not apply to these
    030 IF INF8-10 IS LT C'040' /* Reports and prevents false breaks
    040      GOTO 020.          /* And total prints caused by breaks
    050 CHECKBREAKS             /* In the non-applicable data.
```

CALL

```
CALL user-routine-name [flddef|C'xxx'|P'nnn'|X'...'] ...
```

Term	Description
CALL	Loads a user coded routine at compilation time (VSE) or execution time (MVS) and passes control to that routine each time the executing VISION:Report program passes through the CALL statement. Optionally, CALL passes to the user routine the address of user-specified data areas and literals.
user-routine-name	Enter the name of the user routine to be called. You use the contents of this operand to retrieve the module from the VSE library, sublibrary or appropriate MVS load library and you must therefore match exactly the name under which the routine was cataloged.
flddef C'xxx' P'nnn' X'...'	Enter data area pointers or literals appropriate to the user routines. Literals may be EBCDIC, packed decimal, or hexadecimal. Data area pointers or field definitions can be any addressable VISION:Report area. Data pointers can be coded on one line or continued onto additional lines as needed.

User subroutines that have been link edited with AMODE (31) and RMODE (ANY) are supported subject to the operating system support. For MVS users, this includes z/OS and OS/390. For VSE users, this support is effective starting with VSE/ESA 1.3.

Language Environment Support

VISION: Report supports Language Environment (LE) from IBM. LE provides a common run-time environment and run-time services for all Language Environment conforming programming language products.

VISION:Report is compatible with IBM's Language Environment (LE) and previously known as LE/370, and has been tested with COBOL and Assembler programs. If a user written routing is invoked via a CALL statement in VISION:Report, VISION:Report will automatically determine if it has been with LE or not, and set up the appropriate environment accordingly. If LE, the appropriate run-time libraries must be concatenated accordingly.

Data Area Pointers

Field Definitions

Code a standard VISION:Report field definition for the leftmost position of the field to be referenced by the user routine (for example, if your routine references INF1-5-P, code INF1). At execution time, the current memory address of INF1 is passed to the user routine.

Field length and format values are not passed by VISION:Report. It is your responsibility to consider field length and format in the user routine. All VISION:Report areas can be addressed.

VISION:Report loads called user routines by the LOAD macro or RELOAD macro (VSE) on a dynamic basis. An appropriate message is issued if there is not sufficient space to accommodate the called user routine.

Literals

Term	Description
C'xxx'	Up to a 40-character EBCDIC literal.
P'nnn'	Up to a 10-digit packed literal. In most VISION:Report statements where a packed literal is coded, there is an associated field to provide implicit length for the generated literal. This is not true here and you must code leading zeros and an odd number of digits to ensure generation of a literal of the correct length. To generate a 3-byte packed literal set to 1, you would code P'00001'.
X'...'	A hexadecimal literal of up to 36 bytes.

Note: If you pass a literal to a subroutine, make sure the subroutine does not change the value of the literal. Otherwise, unexpected results can occur.

Literals coded as described above are generated in the VISION:Report static storage. At execution time VISION:Report passes to the user routine the address of the leftmost position of the literal.

As noted under VISION:Report/User Routine Interface, VISION:Report does not pass values to the user routine but does pass the memory addresses where the values are stored.

The following example illustrates this technique:

```
CALL DBOMP DET1 C'CLIENT'
```

- DBOMP is the name of the called routine.
- DET1 is the address where the data area is to return records.
- CLIENT is the name of the file to be read.

Note: A space separates the operands; do not use commas as operand separators.

Using the CALL Statement to Execute User Coded Routines

The CALL statement allows you to call user-written coding. It is designed to be coded and function almost identically to the Assembler and COBOL CALL statements. Some possible uses for called user code are:

- Interface VISION:Report with databases such as TOTAL, IMS, DL/I, CA-IDMS/DB.
- Read or write files not supported by VISION:Report.
- Translate data files to or from ASCII or other presentation.
- Perform functions not within the scope of VISION:Report.

Technical Considerations

- For your convenience, VISION:Report will call user subroutines that are 'above the line' or 31-bit mode if the user subroutine was linked in that mode.
- You can execute any number of user routines in the same VISION:Report run or execution. You can call the same routine in more than one CALL statement; however, it is resident in memory only once.
- You must link edit called code to a VSE executable library (phase) or MVS load library, as appropriate.
- Multiple entry points are not supported.
- Entry to the called code is made at the entry point of the routine. If the entry point has been defined by an Assembler END statement pointing to an address within the module, the LOAD macro or CDLOAD (VSE) provides the entry point address.

- Called code can be Assembler or COBOL, or other languages as long as the code is compatible with the rules and linkages required by our CALL processor. Specifically, FORTRAN and PL/I may not be compatible with the Assembler and COBOL linkage conventions, rules, and VISION:Report requirements. However, you may be able to create Assembler, COBOL, and/or PL/I interfaces that will allow you to use COBOL or PL/I called code.

  In the Optional Material file that was downloaded during the VISION:Report installation, a COBOL interface (ANSINT) can serve as an example. ANSINT is available in source as well as object module. Although this interface has been tested extensively, it may not work in your particular environment.

  Note: With the technology now available, you may no longer need this subroutine.

- User routines can also be referenced by the VISION:Report LOAD statement, that only loads and provides addressability (with PTx), but does not pass control to it.

VISION:Report/User Routine Interface

When VISION:Report transfers to a called user routine, the general register contents are:

Register Number	Contents
Register 14	Contains return address to VISION:Report. You should save this address during execution of the called routine and restore it at the conclusion of the routine. You should return to VISION:Report through Register 14.
Register 15	Contains the entry point to the called user routine. It can be used as a beginning or get started base register. Upon exit, the subroutine may put a return value from 1 to 4095 in Register 15.
Registers 2 through 12	These registers must be saved by the called routine immediately upon entry. These registers are available for use by the called routine and must be restored before returned to VISION:Report.

Register Number	Contents
Register 1	Points to a parameter list of any number of fullwords. The fullwords in turn contain the memory address of all the parameter values that you coded as operands in the CALL statement. For example, the first fullword points to the first parameter following the user routine name and the second fullword points to the second parameter. The leftmost byte of the last parameter contains X'80' as a flag to indicate that it is the last parameter. If no parameters are used, R1 is undefined.
Register 13	Points to an area generated by VISION:Report consisting of 18 fullwords to accomplish register saving and return. Register 13 is set to the address of word 1 of these 18 words. Standard linkage conventions apply to this 18-word area: Word 1: FORTRAN / PL / I use Word 2: backward chain pointer Word 3: forward chain pointer Word 4-18: Registers 14, 15, 0, 1,..., 12

COBOL Considerations

VISION:Report requires subroutines to be cataloged to the phase library / load library prior to being called by VISION:Report.

COBOL Subroutines

Subroutines cannot be cataloged with VISION:Report. For more information, see the consideration as noted on the extracted data from the COBOL Programmer's Guide.

The solution is:

- Prepare and assemble an initialization routine in Assembler. Catalog it to relocatable library / object library.
- Link edit the Assembler routine with the COBOL subroutine into the phase library / load library.

or

- Under MVS, issue a CALL to ILBOSTP0 (or the appropriate module name) prior to issuing any VISION:Report action verb. This establishes the COBOL environment and negates the requirement of ANSINT.

The ILBOSTP0 interface is required for COBOL II. The following is excerpted from the VS COBOLII Application Programming guide:

Calling COBOL from a non-COBOL Program

If the first program in the application is not COBOL, there can be significant degradation if COBOL is repeatedly called, since the COBOL environment must be initialized and terminated each time a COBOL main program is invoked. This overhead can be reduced by using one of the following:

- Call the first program from a COBOL stub program (a program that contains just a CALL statement to the original first program).
- Call ILBOSTP0 from the first program to make it appear as the COBOL main program (this is provided for compatibility with OS/VS COBOL; calling IGZERRE is preferred over calling ILBOSTP0).

The ILBOSTP0 library routine, which was available in OS/VS COBOL to support non-COBOL programs acting as main programs, is still supported. See Overriding the ILBOSTP0 AMODE/RMODE Attributes (in the VS COBOLII Application Programming guide) for more information on ILBOSTP0.

The significance of this method is that it makes an assembler program act like a COBOL main program, so that subsequently called COBOL programs act like subprograms. Also, use of STOP RUN results in control being returned to the caller of the caller of the first COBOL program.

ILBOSTP0 is used to create a reusable run-time environment; the non-COBOL caller will act like the main COBOL program.

ANSINT

The Assembler initialization subroutine, ANSINT (ANS Interface), provides you with an interface to an American National Standard COBOL program. ANSINT has been included as part of the installation process and is available in the object library, as well as in the source/optional material library.

The ANSINT subroutine must be link edited with your COBOL program and must be included first. Care must be taken to ensure that ANSINT is linked ahead of your COBOL program. ANSINT supports DOS/VS and OS/VS COBOL, as well as COBOL II.

Be sure that the proper COBOL object library is in the correct search sequence, as there are duplicate object module names for DOS/VS or OS/VS COBOL and COBOL II.

VSE Considerations

The following example shows how to compile and link edit your COBOL subroutine, CALLRTN1, into the phase library. Observe the sequence of the PHASE and the INCLUDE statements. You must follow the sequence of these two statements.

```
//  OPTION CATAL
    PHASE  CALLRTN1,*
       INCLUDE ANSINT
//  EXEC FCOBOL
... cobol source
/*
// EXEC LNKEDT
```

MVS Considerations

The following example shows how to compile and link edit your COBOL subroutine, CALLRTN1, into the load library. Observe the sequence of the ORDER statement. It is essential to follow this sequence.

```
//COMPILE                   EXEC COBVCL             VS/COBOL
//COB.SYSIN                 DD   *
    .. cobol source
//LKED.OBJLIB    DD  DISP=SHR,
                       DSN=QUIKJOB.OBJ              ANSINT HERE
//LKED.SYSIN     DD  *
      INCLUDE OBJLIB(ANSINT)
      ORDER ANSINT,CALLRTN1
      NAME    CALLRTN1(R)
/*
```

CHECKBREAKS

```
CHECKBREAKS [ON BREAKS PERFORM seq-no THRU seq-no]
```

There are two forms of the CHECKBREAKS statement:

- No operands.
- Accesses a total printing time VISION:Report user routine.

CHECKBREAKS with No Operands

CHECKBREAKS performs the actual comparison and, if total level breaks have occurred, causes the totals to be printed, added to the next level of accumulators, and cleared as appropriate.

If you use a BREAK statement, a CHECKBREAKS statement must be used to obtain desirable results.

If a BREAK statement is specified and no CHECKBREAKS statement is issued during the execution of the program, total breaks do not occur, and the printing of totals at the desired time does not occur.

If VISION:Report reaches EOJ automatically or by a GOTO EOJ statement under these conditions, all totals are printed at end of job.

The values in each level of total are printed and then added as part of the grand total. In other words, the entire accumulation is accumulated in the minor or level one accumulators, printed, added to the level two accumulators, printed, added to the level three accumulators, continued in this sequence, then added to the grand total accumulators, and printed as grand totals.

A BREAK statement with no CHECKBREAKS is useful if you want to read a file and total the dollar values, but do not want any printout except the totals for the entire file.

For example:

```
010 GET
    BREAK 1 INF1-2
    ACCUM INF4-7-P IN A 4 BYTE CTR, ON BREAKS......
    GO TO 010
999 END
```

The entire INF file is read and the data in each record positions 4-7 packed are accumulated in a 4-byte (7-digit) accumulator. EOF for the INF file transfers to VISION:Report internal EOJ, the level 1 or minor accumulator is printed, added to the grand total accumulator, minor accumulator zeroed, and the grand total accumulator is printed.

CHECKBREAKS ON BREAKS PERFORM seq-no THRU seq-no

This form of the CHECKBREAKS statement allows you to specify a PERFORM/EXIT routine. This routine is executed when a control (total) break is recognized and before accumulators are printed, added to the next level, and zeroed.

Note: Only use this form of CHECKBREAKS if you need to gain control at total printing time.

CHECKBREAKS performs the actual comparison of the BREAK fields in the current input record against the comparable fields from the last input record that VISION:Report saved.

If total breaks have occurred, the specified PERFORM/EXIT subroutine is executed once for each level of break that occurred.

VISION:Report accomplishes the following tasks relative to the user routine:

- Values area locations VAL180, VAL181, and VAL182 are created and used as follows:

Location	Description
VAL180	Posts an EBCDIC 1, 2...9 or F before PERFORM xxx through yyy to indicate the level of the break being passed to xxx through yyy. You can evaluate VAL180 if user processing logic differs between levels. If you want your total printing time routine to be repeated, move an EBCDIC 'R' to VAL181. Your printing routine will be reentered until you move any other character to VAL181.
VAL181	Blanks VAL181 before PERFORM xxx through yyy. If your PERFORM logic determines that no totals should be printed, move an EBCDIC 'N' to VAL181 before exiting at yyy. If VISION:Report finds an N at VAL181, space before and space after total printing is suppressed for this occurrence of that break.
VAL182	Blanks VAL182 before PERFORM xxx through yyy. If your PERFORM logic determines that a page eject is appropriate, then move an EBCDIC 'E' to VAL182. If VISION:Report finds an E at VAL182, a page eject and DOHEADERS are executed after total line printing (or suppression through VAL181=N).

VISION:Report increments the addresses of CTA...CTP to point at the accumulators for the level being passed. When VAL180 contains 1, CTA...CTP point at the level 1 (minor) accumulators; when VAL180 contains 2, CTA...CTP point at the level 2 accumulator set, and so on.

For example, if three break levels (minor, intermediate, major) are specified and a third level major break occurs:

- Post '1' to location VAL180, blank VAL181 and VAL182.
- Set addresses to CTA...CTP at level 1 accumulators.
- Perform seq-no through seq-no.
- Test for user modification of VAL181 and VAL182; act accordingly.
- Print level 1 accumulators, add level 1 accumulators to level 2, zero level 1.
- Post '2' to location VAL180, blank VAL181 and VAL182.
- Set addresses of CTA...CTP at level 2 accumulators.
- Perform seq-no through seq-no.
- Test for user modification of VAL181-182; act accordingly.

- Print level 2 accumulators, add level 2 accumulators to level 3, and zero level 2 accumulators.
- Post '3' to location VAL180, blank VAL181-182.
- Set addresses of CTA...CTP at level 3 accumulators.
- Perform seq-no through seq-no.
- Test for user modification of VAL181-182; act accordingly.
- Print level 3 accumulators, add level 3 accumulators to Final level, and zero level 3 accumulators.
- Set addresses of CTA...CTP back to level 1 accumulators.
- Return to statement following CHECKBREAKS.

CLOSE

```
CLOSE {INF|INA-INZ|DET|OFA-OFZ}
```

VISION:Report automatically closes all open files at end of job. However, as a matter of good coding technique, if you open a file, you should close it.

Term	Description
CLOSE	Allows an OPEN statement to reopen a file for additional processing or releases the buffer space for a file to obtain additional memory (MVS only) for other called programs or files.
INA-INZ DET OFA-OFZ	This can be any VISION:Report input and/or output file name. INF is the default.

Example

```
005 GET INC ATEND 100            /* INC opened automatically
    IF INC1 IS NOT EQ C'-'       /* Use only '-' records
    GOTO 005.
        Processing statements
    GOTO 005
100 CLOSE INC                    /* Close INC file
    OPEN INC                     /* Reopen INC file
120 GET INC ATEND EOJ
    IF INC1 IS EQ C'-'           /* Do not use '-' records
    GOTO 120.
        Processing statements
    GOTO 120
```

CLOSER (VSAM ONLY)

```
CLOSER {INF|INA-INZ|DET}
```

VISION:Report automatically closes all open files at end of job. However, as a matter of good coding technique, if you open a file, you should close it.

Term	Description
CLOSER	Performs a CLOSE and an OPEN without releasing control blocks and regenerating them. Resets counters.
INA-INZ DET	This can be any VISION:Report input and/or output file name. INF is the default.

Example

```
005 GET INC ATEND 100            /* INC opened automatically
    IF INC1 IS NOT EQ C'-'       /* Use only '-' records
    GOTO 005.
        Processing statements
    GOTO 005
100 CLOSER INC                   /* Close INC file
120 GET INC ATEND EOJ
    IF INC1 IS EQ C'-'           /* Do not use '-' records
    GOTO 120.
        Processing statements
    GOTO 120
```

CONDATE

```
CONDATE from-mask AT flddef1 TO to-mask AT flddef2
```

Term	Description
CONDATE	Maintains a 6-digit date in a 2-byte binary field. The date is compressed into two bytes based upon a mask pattern. You must use the EXDATE statement to restore the date to 6 digits. Century is not indicated in CONDATE and EXDATE format. However, you can still use it.
from-mask	This 6-character mask represents the format of the date to be compressed. The mask must be one of the following formats: MMDDYY Month(MM), Day(DD), Year(YY) DDMMYY Day(DD), Month(MM), Year(YY) YYMMDD Year(YY), Month(MM), Day(DD) YYDDMM Year(YY), Day(DD), Month(MM)
flddef1	This 6-character VISION:Report area contains the date to be compressed. The date must be in the same pattern as the from-mask. Otherwise invalid results occur. Example: From-Mask Date DDMMYY 310301 (31 March, 2001)
to-mask	This 16-character mask indicates what each bit in the 2-byte VISION:Report field is to represent. The 2-byte field contains the compressed date. The month, day, and year of the date require the following: MMMM 4 bits for the month DDDDD 5 bits for the day YYYYYYY 7 bits for the year The mask must be in one of the following formats: MMMMDDDDDYYYYYYY Month, day, year DDDDDMMMMYYYYYYY Day, month, year YYYYYYYMMMMDDDDD Year, month, day YYYYYYYDDDDDMMMM Year, day, month Example:To-Mask Date DDDDDMMMMYYYYYYY 310396 (31 March, 96)

Term	Description
flddef2	This 2-byte binary VISION:Report field contains the date after it has been compressed.

DELETE

```
DELETE {INA-INZ|DET} [ONERROR seq-no]
```

Term	Description
DELETE	Deletes the last record retrieved from a KSDS or RRDS VSAM file. **Note:** This statement cannot be used with ESDS VSAM files.
INA-INZ DET	File name can be any VISION:Report input file name (INA-INZ, DET).
ONERROR seq-no	Causes VISION:Report to automatically transfer to the statement the sequence number when a VSAM error occurs. This allows you to examine the error codes (RC/EC) and take appropriate action. If this operand is not coded and a VSAM error occurs, execution continues with the next statement. It then becomes your responsibility to check for these errors.

DISPLAY

```
DISPLAY {flddef|C'xxx'}
```

Term	Description
DISPLAY	Displays the specified field or literal on the system console.
flddef C'xxx'	Defines the field to be displayed, up to 80 positions. The data displayed is presumed to be EBCDIC; no data translation is made. If you specify a character literal, the length can be up to 64 characters.

Example

```
010 DISPLAY C'ENTER MONTH TO PROCESS AS NN'      /* Notify operator
    020 ACCEPT WST1-2                            /* Accept operator reply
    030 IF WST1-2 IS NUMERIC                     /* Test reply for numeric or
    040      GO TO ...                           /* Go to the processing routine.
```

DIVD

```
DIVD flddef1 nD BY {flddef2|C'xxx'|P'nnn'|X'...'} nD GIVING flddef3 nD[R]
```

Term	Description
DIVD	Divides the contents of the dividend field by the contents of the divisor field or literal and stores the answer in the quotient field. Decimal alignment specifications are required for all operands. The remainder is placed at VAL30-37-P in packed decimal format. Fields and literals in any data format may be divided by fields in any other data format. Automatic data conversion is performed in all cases.
flddef1	Defines the dividend field in the standard format. Data may be in EBCDIC format (maximum of 19 digits), packed format (maximum of 10 bytes), or binary format (maximum of 8 bytes).
nD	Number of decimal positions in the dividend field (for example, if the dividend field has 2 decimal positions, code 2D).
BY	Required noise word.
flddef2 C'xxx' P'nnn' X'...'	Defines the divisor field in standard format with the same length limitations as dividend field. You can code a constant as the divisor with a maximum of 11 digits in the constant.
nD	Number of decimal positions in divisor flddef or literal.
GIVING	Required noise word.
flddef3	Defines the quotient field in standard format with the same length limitations as dividend field.
nD	Number of decimal positions in quotient field.
R	Rounds the quotient.

All division and multiplication is performed internally in packed decimal. Dividends, divisors, and quotients may be up to 10 bytes packed, 19 bytes EBCDIC, and 8 bytes binary.

Quotients may have 10-digits with a maximum value of 2,147,483,647. An attempt to compute a binary quotient greater than 2,147,483,647 results in a FIXED POINT DIVIDE exception.

DOHEADERS

```
DOHEADERS [PAGEONE]
```

Term	Description
DOHEADERS	Ejects to the next page and prints page headers. This statement allows you to force page headers at your convenience.
PAGEONE	This entry forces page headers to restart with page number 1. If PAGEONE is not specified, the forced page headers continue with the next page number.

DROP

```
DROP user-routine-name
```

Term	Description
DROP	Deletes called and/or loaded modules from memory. Use this statement if you intend to eliminate a module from memory in order to make room for other modules. VISION:Report automatically releases all modules at EOJ.
user-routine name	This name must match the name under which the routine was called or loaded. An error occurs if the module specified here was never coded on a LOAD or a CALL statement.

VISION:Report ignores an attempt to drop a module that is not in memory at that time.

Example

```
010 CALL MYMODULE WST1 INF1
    .
    .
    .
    DROP MYMODULE
```

EJECT

```
EJECT
```

This statement is declarative and therefore cannot be the subject of a transfer in control statement (for example, GOTO, PERFORM).

Term	Description
EJECT	Forces page ejection of the VISION:Report statement listing. Different routines can be printed on different pages, thus making the VISION:Report listing easier to read.

END

```
END
```

Term	Description
END	Indicates the end of VISION:Report statements.

In MVS, the END statement is only required if a user table is being read, but as a good practice, include an END statement in any program.

In VSE, if the input file is in 80 byte form, it should start immediately after this statement and end with the appropriate data statement. If a user table and 80-byte input are to be read, see the examples below.

80-Byte Input Only

```
INFCARD
010 GET INF                       /* Read the input record
    MOVE INF1-80 to PRT1          /* Move record to printer
    PRINT                         /* Print the record
    GO TO 010
999 END
    User input data for INF file goes here
/*                                /* VSE delimiter
/&
```

80-Byte Input and Table Input

VSE Example

```
 INFCARD
    TABLSPEC 0200 01 05                 /* Expect 200 table entries
020 GET                                 /* Get an input record
    IF INF1-5 IS ONTABLE                /* If number is on the table
         MOVE INF1-80 to OFA1-80        /* Move the record to the output
         WRITE OFA.                     /* Write the record to output
    GO TO 020                           /* Go get next input record
999 END
    Table data goes here
/*
    INF data statements go here
/*
/&
```

MVS Example

```
//SYSIN DD *
010 TABLSPEC 0200 01 05
020 GET
030 IF INF1-5 IS ONTABLE
040      MOVE INF1-80 TO OFA1-80
050      WRITE OFA.
060 GOTO 020
999 END
    Table data goes here
/*
```

EQU

```
EQU data-name {flddef|(n)|(x)} [nC|nE|nN]
[C'xxx'|P'nnn'|X'...'|ZERO|BLANK|SPACE|HIVALUE|LOVALUE]
```

Term	Description
EQU	Equates a data name to a VISION:Report field definition. It allows you to code descriptive data names in VISION:Report statements. VISION:Report interprets these as the field definition supplied as a function in the EQU statement. Constants (or literals) can be specified optionally. EQU statements must precede VISION:Report imperative statements. You should never use VISION:Report field definitions (for example, @VAL...) as data names. If you use an equated data name, the name must be at least three characters long or you can get unpredictable results, especially with MOVE and/or MOVE VARIABLE LENGTH.
data-name	Code the data name. The data name can be up to 34 characters long and consist of any string of characters in the EBCDIC set. (Parentheses and Embedded blanks are not allowed when using the REPORT statement.) The first blank encountered terminates the data name. Do not use the following characters as part of the data name: ' apostrophe " double quotation mark (left parenthesis) right parenthesis : colon , comma ; semicolon * asterisk / slash . period ? question mark + plus sign ¬ not sign ! exclamation point | vertical bar & ampersand < less than > greater than { left brace } right brace [left bracket] right bracket You can use the hyphen (-) sign or underscore (_), but not as the last character of a data name.
flddef	After allowing one or more spaces to terminate the data name, code the VISION:Report field definition that applies to the data name. This definition should consist of the AREA/FROM/TO format of the field.
n	If there is a previously defined field definition, only the field length needs to be coded. The equated names are assigned adjacent positions (based on the stated length) in the same area as the last equate.

Term	Description
(x….x)	Redefine the name. If there is a previously defined field definition, only the field name needs to be coded.
nC nE nN	For use with MOVE equated name to PRT area and/or REPORT statement. Edit field specifications for printing may be stated here rather than in each MOVE statement where the equated name is used. If the MOVE statement includes edit specifications, the EQU edit specifications are overridden. (See MOVE (Expanded Editing) for more coding details.) A non-blank value indicates this is a numeric field. The length of a character field is 19 if this field is used.
C'xxx' P'nnn' X'...'	Literal or figurative constants stated here are moved into the field definition prior to execution. Literals may not exceed the size of the equated field area or a diagnostic occurs. Character literals may contain up to 40 characters. Packed literals may be up to 19 digits long. Hexadecimal literals may be from 2 to 24 characters long.
ZERO BLANK SPACE HIVALUE LOVALUE	Figurative constants (for example, ZERO, BLANK) may be initialized into an equated data area. SPACE and BLANK may be used only in conjunction with EBCDIC fields. LOVALUES and HIVALUES are not usable within packed field equates. See General Rules, Rule L for more information about figurative constants.

The equated field format determines the actual format of the equated name. Hexadecimal literals are moved 2 characters at a time into 1 byte of the equated area. Formats must be the same or a combination of hexadecimal with EBCDIC.

If you try to initialize a file area, an error message will be issued.

Examples

These equates are assigned to:

Equates assigned to

```
EQU                MY-RECORD           WST1-66             /*GROUP AREA
EQU                MY-RECORD                               /*REDEFINES
EQU                NAME                WST1-30             /*WST1-30
EQU                ADDRESS             (15)                /*WST31-45
EQU                CITY                (10)                /*WST46-55
EQU                STATE               (2)                 /*WST56-57
EQU                ZIP                 (9)                 /*WST58-66
```

Another use of 'n' can be employed for group area designator and elementary field definition redefines (much like COBOL redefines). Equated names may overlap or redefine a previously equated data area.

Equated names

```
EQU    WORK-AREA1      WST             /*WST00-00      Group area designators**
EQU    TAX             (4)-P           /*WST1-4-P
EQU    PRCT-WHLD       (4)-P           /*WST5-8-P
EQU    COMMPRCT        (4)-P           /*WST9-12-P
EQU    WORK-AREA2      WST             /*WST00-00     Group area designators**
EQU    TAX2            (TAX)           /*WST1-4-P     Redefine TAX
EQU    PRCT2           (PRCT-WHLD)     /*WST5-8-P     Redefine
EQU    COMM2           (4)             /*WST9-12       Redefine
```

**When using a group area designator, you cannot use edit field specifications, literals, and figurative constants, and you can never directly reference the equated name in any VISION:Report statement.

You can perform this type of equate with edit field specifications, literal, and figurative constants; however, you should use them with care. The operands coded in an EQU statement are applied when that equated name is assigned.

Each literal or figurative constant is moved into the equated data area. In the final occurrence of an EQU statement with an area that has been previously equated, the literal or figurative constant is the data to reside in the equated area upon execution.

If you use a literal or figurative constant in a file area, such as INF, an error message is issued.

These equates are assigned to:

Equates assigned to			
EQU	NAME	WST1-20	/* WST1-20
EQU	ADDRESS	(20)	/* WST21-40
EQU	BILL-AMOUNT	(4)-P	/* WST41-44-P
EQU	BILL-DATE	(6)	/* WST45-50
EQU	CUSTOMER-CODE	(2)-B	/* WST51-52-B
EQU	NAME-PR	PRT1-20	/* PRT1-20
EQU	ADDRESS-PR	(20)	/* PRT21-40
EQU	BILL-AMOUNT-PR	(8) 2C	/* PRT41-48
EQU	UNIT-PRICE	INF1-3-P	
EQU	UNIT-COST	INF7-9-P	
EQU	PR-UNIT-PRICE	PRT10	
EQU	SUBTRACT-FIVE	INF10	
EQU	DATE1	WST70-79	C'91029xxxx'
EQU	MY-CTR	WST90-93-P	ZEROS
EQU	IN-DATE	INZ9995-10000	/* Format MMDDYY

```
050 MOVE UNIT-PRICE TO PR-UNIT-PRICE 2C
```

is interpreted as

```
050 MOVE INF1-3-P TO PRT10 2C
```

EQU Statements for VAL Area

A table of EQU statements for the VAL area is loaded during the compile phase. Any of these data names can be used in VISION:Report programs without coding EQU statements. See the section VAL Area in Chapter 2 for a more detailed description.

Change these equates by selecting the source module, QUIKVEQU (MVS) or QUKBVEQU (VSE), from the optional material file. Refer to the VISION:Report Installation Guide for information about Optional Material.

The EQU statements as supplied are:

Field Data Name	Definition	Edit Spec
@VAL-AREA	VAL0001-0799	
@VAL-UNDEF-LTH	VAL0001-0004-P	0C
@VAL-IPL-DATE	VAL0005-0012	
@VAL-IPL-MONTH	VAL0013-0021	
@VAL-DOS-UPSI	VAL0022-0029	
@VAL-REMAINDER	VAL0030-0037-P	0C
@VAL-JOBNAME	VAL0038-0045	
@VAL-RETURN-CD	VAL0046-0049	
@VAL-MMDDYY	VAL0050-0055	D
@VAL-IPL-MM	VAL0050-0051	
@VAL-IPL-DD	VAL0052-0053	
@VAL-IPL-YY	VAL0054-0055	
@VAL-YYMMDD	VAL0056-0061	D
@VAL-TIME	VAL0062-0066	
@VAL-ABEND-CD	VAL0067-0070	
@VAL-YYDDD	VAL0071-0075	
@VAL-PARMBYTES	VAL0076-0077-B	0C
@VAL-PARAMETER	VAL0078-0177	
@VAL-COMREG	VAL0078-0088	
@VAL-BREAK-LVL	VAL0180-0180	
@VAL-BREAK-PRT	VAL0181-0181	
@VAL-BREAK-EJC	VAL0182-0182	
@VAL-INF-EOF	VAL0196-0196	
@VAL-DET-EOF	VAL0197-0197	
@VAL-INC-EOF	VAL0198-0198	
@VAL-IND-EOF	VAL0199-0199	
@VAL-SORT-EOF	VAL0200-0200	
@VAL-SORT-LTH	VAL0201-0204-B	0C

Field Data Name	Definition	Edit Spec
@VAL-PHASE-LTH	VAL0205-0208-B	0C
@VAL-WHEN-LTH	VAL0225-0228-B	0C
@VAL-FILE-NAME	VAL0229-0231	
@VAL-REC-COUNT	VAL0232-0235-B	0C
@VAL-DTF-DCB	VAL0236-0239-B	
@VAL-VSAM-FILE	VAL0240-0240	
@VAL-VSAMACCES	VAL0241-0241	
@VAL-VSAMLRECL	VAL0243-0246-B	0C
@VAL-VSAM-RC	VAL0247-0247-B	
@VAL-VSAM-EC	VAL0248-0248-B	
@VAL-VSAM-RCEC	VAL0247-0248-B	
@VAL-VSAM-RBA	VAL0249-0252-B	
@VAL-VSAM-ERR	VAL0253-0255	
@VAL-VSAM-RBX	VAL0249-0256-B	(for VSAM-XRBA only)
@VAL-VSAM-ERX	VAL0257-0259	(for VSAM-XRBA only)
@VAL-RESV-MEM	VAL0257-0260-B	
@VAL-SQLCA	VAL0273-0408	
@VAL-SQL-CAID	VAL0273-0280	
@VAL-SQL-LEN	VAL0281-0284-B	0C
@VAL-SQL-CODE	VAL0285-0288-B	
@VAL-SQL-ERRML	VAL0289-0290-B	
@VAL-SQL-ERRMC	VAL0291-0360	
@VAL-SQL-ERRP	VAL0361-0368	
@VAL-SQL-ERRD1	VAL0369-0372-B	
@VAL-SQL-ERRD2	VAL0373-0376-B	
@VAL-SQL-ERRD3	VAL0377-0380-B	
@VAL-SQL-ERRD4	VAL0381-0384-B	
@VAL-SQL-ERRD5	VAL0385-0388-B	
@VAL-SQL-ERRD6	VAL0389-0392-B	
@VAL-SQL-WARN0	VAL0393-0393	

Field Data Name	Definition	Edit Spec
@VAL-SQL-WARN1	VAL0394-0394	
@VAL-SQL-WARN2	VAL0395-0395	
@VAL-SQL-WARN3	VAL0396-0396	
@VAL-SQL-WARN4	VAL0397-0397	
@VAL-SQL-WARN5	VAL0398-0398	
@VAL-SQL-WARN6	VAL0399-0399	
@VAL-SQL-WARN7	VAL0400-0400	
@VAL-SQL-WARN8	VAL0401-0401	
@VAL-SQL-WARN9	VAL0402-0402	
@VAL-SQL-WARNA	VAL0403-0403	
@VAL-SQL-STATE	VAL0404-0408	
@VAL-SQL-LSTFN	VAL0409-0410-B	
@VAL-YYYY	VAL0411-0414	
@VAL-INA-EOF	VAL0415-0415	
@VAL-INB-EOF	VAL0416-0416	
@VAL-INE-EOF	VAL0417-0417	
@VAL-ING-EOF	VAL0418-0418	
@VAL-INH-EOF	VAL0419-0419	
@VAL-INI-EOF	VAL0420-0420	
@VAL-INJ-EOF	VAL0421-0421	
@VAL-INK-EOF	VAL0422-0422	
@VAL-INL-EOF	VAL0423-0423	
@VAL-INM-EOF	VAL0424-0424	
@VAL-INN-EOF	VAL0425-0425	
@VAL-INO-EOF	VAL0426-0426	
@VAL-INP-EOF	VAL0427-0427	
@VAL-INQ-EOF	VAL0428-0428	
@VAL-INR-EOF	VAL0429-0429	
@VAL-INS-EOF	VAL0430-0430	
@VAL-INT-EOF	VAL0431-0431	

Field Data Name	Definition	Edit Spec
@VAL-INU-EOF	VAL0432-0432	
@VAL-INV-EOF	VAL0433-0433	
@VAL-INW-EOF	VAL0434-0434	
@VAL-INX-EOF	VAL0435-0435	
@VAL-INY-EOF	VAL0436-0436	
@VAL-INZ-EOF	VAL0437-0437	
@VAL-SQL-MLEN	VAL0438-0439-B	
@VAL-SQL-MALL	VAL0440-0759	
@VAL-SQL-M1	VAL0440-0519	
@VAL-SQL-M2	VAL0520-0599	
@VAL-SQL-M3	VAL0600-0679	
@VAL-SQL-M4	VAL0680-0759	
@VAL-EXP-RSVR1	VAL0760-0799	

EXDATE

```
EXDATE from-mask AT flddef1 TO to-mask AT flddef2
```

Term	Description
EXDATE	Decompresses a 2-byte binary date, created by a CONDATE statement, into a six-character date. Century is not indicated in CONDATE and EXDATE format. However, you can still use it.
from-mask	■ This 16-character mask indicates what each bit in the 2-byte VISION:Report field represents. The month, day, and year of the date require the following mask: MMMM 4 bits for the month DDDDD 5 bits for the day YYYYYYY 7 bits for the year ■ The mask must be in one of the following formats and must be the same as the to-mask used in the EXDATE statement for this field. MMMMDDDDDYYYYYYY Month, day, year DDDDDMMMMYYYYYYY Day, month, year YYYYYYYMMMMDDDDD Year, month, day YYYYYYYDDDDDMMMM Year, day, month
AT	Required noise word.
flddef1	This two-character VISION:Report field contains the compressed date. For example: To-Mask MMMMDDDDDYYYYYY Condensed Date 310396 (31 March, 96)
TO	Required noise word.
to-mask	This 6-character mask describes the format of the decompressed data. This mask is normally the same as the from-mask used in the CONDATE statement for this field. The mask must be in one of the following formats: MMDDYY Month(MM), Day(DD), Year(YY) DDMMYY Day(DD), Month(MM), Year(YY) YYMMDD Year(YY), Month(MM), Day(DD) YYDDMM Year(YY), Day(DD), Month(MM)

Term	Description
AT	Required noise word.
flddef2	This is the 6-character VISION:Report field into which the date is expanded. The date is in the format specified by the to-mask.

EXIT

```
seq-no EXIT
```

Term	Description
EXIT	Represents the end of a performed subroutine. If the statements preceding the EXIT are fallen into (as opposed to performed), the EXIT is deactivated and control transfers to the statement following the EXIT.

There must be a sequence number on the EXIT statement.

GET

```
GET {INF|INA-INZ|DET} [ATEND {seq-no|EOJ}]
```

Term	Description
GET	Reads the next logical record available in the data area specified.
INA-INZ DET	Specifies the data area to be read. If left blank, INF is assumed and you cannot use the ATEND operand. When EOF is reached, the data area for the specified file is filled with high-values (X'FF'). An EBCDIC 'E' (X'C5') is moved to one of the following positions in the VAL area, depending on the file name: VAL196 INF VAL420 INI VAL429 INR VAL197 DET VAL421 INJ VAL430 INS VAL198 INC VAL422 INK VAL431 INT VAL199 IND VAL423 INL VAL432 INU VAL415 INA VAL424 INM VAL433 INV VAL416 INB VAL425 INN VAL434 INW VAL417 INE VAL426 INO VAL435 INX VAL418 ING VAL427 INP VAL436 INY VAL419 INH VAL428 INQ VAL437 INZ

If EOF occurs for a file and INF is the only input file, you have the following options:

- If you do not want control at EOF, do not use the ATEND statement or the GET statement with an ATEND operand. VISION:Report automatically takes final totals and goes to EOJ when EOF occurs.

  ```
  010 GET INC
  ```

- If you do want control at EOF, use either the ATEND statement or the GET statement with an ATEND operand.

  ```
  010 GET DET ATEND 100
     Processing statements
      GOTO 010
  100 DOHEADERS
     Processing statements
      GOTO EOJ
  ```

  If EOF occurs for a file and the ATEND operand is given, VISION:Report automatically transfers to the statement whose sequence number is nnnn or EOJ.

- You can check for EOF in a file by checking the VAL area or checking for high-values in the file's data area.

```
IF VAL197 IS EQ C'E'                    /* DET at EOF?
     IF VAL196 IS EQ C'E'               /* INF at EOF?
     IF VAL199 IS EQ C'E'               /* IND at EOF?
     GOTO 500.                          /* Where 500 is EOJ routine
```

If there are no user high-value records in the files, you can use the following format:

```
IF INF1-5 IS EQ INC1-5                  /* INF eq INC
     IF INF1-5 IS EQ DET1-5             /* INF eq DET
     IF INF1-5 IS HIVALUE               /* And HIVALUE
     GOTO 500.                          /* Where 500 is EOJ routine
```

Term	Description
ATEND	If this operand is used, the sequence number must specify a transfer point to receive control when EOF for the file occurs. If the ATEND operand is used, you must include a GOTO EOJ statement in the EOJ routine to allow VISION:Report, for example, to close all files, take final totals.
seq-no	When EOF occurs, VISION:Report: ■ Moves high-values to the file's record area. ■ Moves an E to the appropriate position in the VAL area. ■ Transfers control to the statement specified by seq-no.
EOJ	Transfers to the end of job routine.

If the input file is a VSAM file, the length of the record is returned in the 2-byte or 4-byte length field preceding the I/O area. See the techniques discussed in the section VSAM Recommendations in Chapter 2 to access this field. When a VSAM file reaches EOF using GET and you want to continue processing (using GET), to avoid error 212 you must close and open the file.

When you specify more than one input file or the input file is not INF, you must recognize EOF with one of the following methods:

- Using the ATEND operand on the GET statement.
- Determining when any or all inputs have reached EOF (checking for high-values in the input area, or testing for E in VAL area).

GOTO

```
GOTO {seq-no|EOJ}
```

Term	Description
GOTO	Transfers control to the statement specified by sequence number of EOJ. You can also specify this as GO TO.
seq-no	Transfers to this statement sequence number.
EOJ	Transfers to the end-of-job routine. This statement has the effect of putting HIVALUES into the INF area, forcing all total levels, and going to end of job after closing out all files.

The GOTO EOJ statement can be very useful in going to early end of job when you have processed all pertinent data in a large file.

For example, if a report being produced applies only to PLANT 07 (INF1-2), close out the run when the first record with a plant number higher than 07 is read.

```
IF INF1-2 IS GT C'07'
     GOTO EOJ.
```

If you want to run a few pages of a report and terminate the run for testing, you can code:

```
EQU PAGE-NR PNR1-4-P
IF PAGE-NR IS GT P'nn'
     GO TO EOJ.
```

HDR

```
HDR {1-6} {[A{carriage control}header data] | [{B-Z}header data]}
```

The HDR statement can only have a sequence number of 1-4 digits.

Term	Description
HDR	Specifies user report header lines, a 120 to nnnn character print line. Must be input in columns 5-7 even if you specify option SEQCHK=NO. (The optional sequence number is in columns 1-4.)
1-6	Identifies the header line number (for example, 1 is the first line, 2 is the second line). You are allowed six heading line numbers, and 26 heading line segments within each heading number. Each statement represents one part of the header print line. Must be input in column 9.
A-Z	Identifies heading line segment (A to Z) within the header line number. This operand must be input in column 10.
carriage control	Specifies the ASA standard carriage control for the A heading line. The carriage control is defined in column 12 and the data to appear in the print line starts in column 13. Blank — Single-space before printing 0 — Double-space before printing - — Triple-space before printing + — Do not space before printing 1 — Skip to channel 1 before printing 2-9, A, B, C — Skip to respective channel before printing
header data	Defines the data to appear in the print line. The A heading line contains the carriage control character and the first 66 print positions (for a total of 67 print positions); the B through Z segments contain up to 66 print positions. Heading line segments may be specified in any order. If only segments A and G are specified, segments B-F default to blanks. For example, a 1B would be part of the first header with printing starting in print position 67. The maximum number of print positions for one line is 1,717 or 67+(25*66). For heading lines B through Z, the header data must start in column 12 and end in column 77. Heading line A uses positions 12-78 (column 12 contains carriage control; the header data starts in column 13). If you use all 66 print positions of heading line A, the 66th position would extend past the STMTEND position. You may have to override the STMTEND option (see OPTION).

Example

Col 5-7	Col 9-10	Col 12-n
HDR	1A	First heading line of report run on $IPLDAT$ page PG. The first character in column 12 is the ASA carriage control character. (The date and page number may be used on any header line.)
HDR	1B-1Z	66 bytes of first heading as a continuation from previous statement.
HDR	2A	Left side of second heading line.
HDR	2B-2Z	66 bytes of first heading as a continuation from previous statement.
HDR	3A	Left side of third heading line.
HDR	3B-3Z	66 bytes of first heading as a continuation from previous statement.
HDR	4A	Left side of fourth heading line.
HDR	4B-4Z	66 bytes of first heading as a continuation from previous statement.
HDR	5A	Left side of fifth heading line.
HDR	5B-5Z	66 bytes of first heading as a continuation from previous statement.
HDR	6A	Left side of sixth heading line.
HDR	6B-6Z	66 bytes of first heading as a continuation from previous statement.

When all headers have been loaded, VISION:Report scans the headers and reacts as indicated:

Term	Description
$IPLDAT$	The 8-byte MM/DD/YY date is placed in the header positions occupied by the constant. This is the date of the system IPL, unless overridden by the // DATE JCL statement in VSE.
$IPLDYYYY$	The 10-byte MM/DD/YYYY date is placed in the header positions occupied by the constant. This is the date of the system IPL, unless overridden by the // DATE JCL statement in VSE.

Term	Description
$DATE$	If this constant is found, VISION:Report places the six-position Julian (YY.DDD) date in the header positions occupied by the constant. This is the date of the system IPL and cannot be overridden by the / / DATE JCL statement in VSE.
$JDYYYY$	If this constant is found, VISION:Report places the eight-position Julian date (YYYY.DDD) in the header positions occupied by the constant. This is the date of the system IPL and cannot be overridden by the / / DATE JCL statement in VSE.
PG	If this constant is found, VISION:Report prints a 4-digit, zero-suppressed page number in the positions where the constant appears. You can check or modify the current page number by using the PNR1-4-P dataname.
PAG	If this constant is found, VISION:Report prints a 5-digit, zero-suppressed page number in the positions where the constant appears. You can check or modify the current page number by using the PNR1-4-P dataname.
$PAGE$	If this constant is found, VISION:Report prints a 6-digit, zero-suppressed page number in the positions where the constant appears. You can check or modify the current page number by using the PNR1-4-P dataname.
$PAGES$	Same as $PAGE$, except VISION:Report prints a 7-digit, zero-suppressed page number.
$JOBNAM$	If this constant is found, VISION:Report places the eight-character job name in the header positions occupied by the constant.
TIM	If this constant is found, VISION:Report places the 5-digit current time (HH.MM) in the header position occupied by the constant. Additionally, you can specify valid equated data names or field definitions enclosed in dollar signs (reserved header names) to place user data fields in the header areas each time page headers are printed. The reserved words ($datanames$) are expanded or compressed as needed to correspond to the data field length. This feature is intended to simplify those cases where sections of a report are identified by file data placed in the page titles. This data should be moved from the record input area to a WST work area so that it can be accessed at end of file time.

Page Header Modification

1. You can reference page headings. Headers 1 through 6 are areas HDA through HDF and are addressable as such. HDA1 is the first visible print position.
2. Page headings are not printed until the first PRINT statement is executed. You can modify the headings with first data record information or information typed in through an ACCEPT statement, as an example.

Example

```
MOVE INF1-10 TO HDF1-10
```

The previous statement moves the input record INF positions 1 through 10 into the sixth heading, positions 1 through 10. Data from an input area should not be used at end of file time.

The PRTSIZE=nnnn option actually determines the maximum length of the header statements. User statement references to PRT are allowed up to the maximum PRTSIZE option value in effect for each respective VISION:Report program. Any value between 121 and the maximum (as defined by the hardware and the operating system) is valid, with 133 as the default.

HEXCOND

```
HEXCOND flddef1 WITH flddef2 { flddef3 | X'...' }
```

Term	Description
HEXCOND	Translates and tests a data field using a 256-byte substitution list with non-zero function bytes in all positions except for the character values A-F and 0-9. The value of each byte in the source is added to the list address and the function byte at the resulting address is inspected for a value of zero. The source field remains unchanged. The bytes in the source field are examined one-by-one, from left to right, until a non-zero function byte is encountered or all the bytes in the source field have been examined. The results of the operation are returned in VAL224-B, VAL225-228-B and PTR. If no non-zero function bytes were encountered or if the length value in the third operand is zero, then VAL224-B and VAL225-228-B are both zero and PTR is unchanged. The character values are then converted into the binary field equivalent in the second operand. If a non-zero function byte is encountered, then it is returned in VAL224-B, the number of bytes scanned is returned in VAL225-228-B, and PTR points to the byte in the source that resulted in the non-zero function byte. If the length value in the third operand is not a multiple of 2, a value of X'FFFFFFFF' is returned in VAL225-228-B.
flddef1	Source data to be converted.
TO	Required noise word.
flddef2	Converted data.
flddef3 X'...'	Optional. A 2 or 4-byte binary field indicating the number of bytes to be converted. If this operand is omitted, then the length of the source field is used.

Example

```
EQU FILLER       WST0
EQU FLDA        (272)
EQU FLDA
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FILLER       (16)  C'0123456789ABCDEF'
EQU FLDB         (200)
    HEXCOND FLDA TO FLDB
    PRINTHEX FLDB

              WST273-472

  02468ACE02468ACE02468ACE02468ACE02468ACE02468ACE02468ACE02468ACE02468ACE02468ACE02468ACE02468ACE0246
  13579BDF13579BDF13579BDF13579BDF13579BDF13579BDF13579BDF13579BDF13579BDF13579BDF13579BDF13579BDF1357
 01..05...10...15...20...25...30...35...40...45...50...55...60...65...70...75...80...85...90...95..100

  8ACE02468ACE02468ACE02468ACE02468ACE0000000000000000000000000000000000000000000000000000000000000000
  9BDF13579BDF13579BDF13579BDF13579BDF0000000000000000000000000000000000000000000000000000000000000000
101..05...10...15...20...25...30...35...40...45...50...55...60...65...70...75...80...85...90...95..100
```

HEXEXPD

```
HEXEXPD flddef1 WITH flddef2 { flddef3 | X'...' }
```

Term	Description
HEXEXPD	Each byte of the binary data in the source is converted into a 2-byte character equivalent in the target. The source field remains unchanged.
flddef1	Source data to be converted.
TO	Required noise word.
flddef2	Converted data.
flddef3 X'...'	Optional. A 2 or 4-byte binary field indicating the number of bytes to be translated. If this operand is omitted, then the length of the source field is used.

Example

```
EQU FILLER       WST0
EQU FLDA         (18)
EQU FLDA
EQU FILLER       (16)  X'0123456789ABCDEF0123456789ABCDEF'
EQU FILLER       (16)  X'0123456789ABCDEF0123456789ABCDEF'
EQU FLDB         (100)
    HEXEXPD FLDA TO FLDB
    PRINTHEX FLDB


 WST33-132 0123456789ABCDEF0123456789ABCDEF0123

FFFFFFFFFFCCCCCCFFFFFFFFFFCCCCCCFFFF00000000000000000000000000000000000000000000000000000000000000000000000000000000

01234567891234560123456789123456012300000000000000000000000000000000000000000000000000000000000000000000000000000000

01..05...10...15...20...25...30...35...40...45...50...55...60...65...70...75...80...85...90...95..100
```

IF

Note: This is the simple IF statement. Note that AND/OR logic requires the IF clause at the beginning of each sentence. See General Rules, Rule K for more information about true condition processing, regarding the usage of a period. (For the compound IF statement, see IF (Compound).

```
IF flddef1 [IS] [NOT]
{EQ|LT|GT|ONTABLE|ALPHA|NUMERIC|ZERO|BLANK|SPACE|HIVALUE|LOVALUE} [TO]
{flddef2|C'xxx'|P'nnn'|X'...'|ZERO|BLANK|SPACE|HIVALUE|LOVALUE} {AND|OR}
```

Term	Description
IF	Compares the contents of a flddef1 against flddef2, a literal, or a constant. If the relationship or condition stated is found to be true, continue processing with the next statement. If the relationship is found to be false, transfer to the next sentence (statement following true processing). (See General Rules, Rule K for more information about true condition processing.)
flddef1	Comparison field. Define the area/field flddef to be compared. See the relationship and literal operands for length limits.
IS	Required noise word.
NOT	The relationship sought is reversed.

Term	Description	
condition	Enter the relationship that establishes a true condition (see General Rules, Rule L). ZERO, BLANK, SPACE, HIVALUE, and LOVALUE are limited to the size of the comparison field.	
	EQ	The comparison field is equal to flddef2.
	LT	The comparison field is less than flddef2.
	GE	The comparison field is greater than or equal to flddef2.
	GT	The comparison field is greater than flddef2.
	ONTABLE	The comparison field is present, as an argument, in the user table. If functions for each argument are in the table, a match against the table causes the function to move to the FUN area. ALPHA
	NUMERIC	The comparison field is all numeric, except the low order position which may have a sign. If the comparison field is packed, it is unpacked into a VISION:Report work area and the compare is made against that work area. See Numeric Comparison.
	ZERO	The comparison field is all zeros.
	BLANK	The comparison field is all blanks.
	SPACE	The comparison field is all spaces.
	HIVALUE	The comparison field is all hex FF.
	LOVALUE	The comparison field is all hex 00.
TO	Required noise word.	

Term	Description	
flddef2 C'xxx' P'nnn' X'...' ZERO BLANK SPACE HIVALUE LOVALUE	An area/field flddef against which the comparison field is to be compared. This field does not have to be the same length and format as the comparison field. Up to 2-gigabyte characters on EBCDIC fields and up to 10 bytes on packed fields.	
	Literal field. The following literals may be used only when EQ, GT, or LT is specified. Enter:	
	C'xxxx'	An EBCDIC literal up to 25 characters long.
	P'xxxx'	A packed literal up to 10 bytes long.
	X'xxxx'	A hexadecimal literal up to 12 bytes long. All entries must be in the range 0 through F.
	The following figurative constants can only be used when EQ, LT, or GT is specified as the relationship operand and limited only by field size:	
	SPACE	The comparison field is all spaces.
	BLANK	The comparison field is all blank.
	ZERO	The comparison field is all zeros.
	HIVALUE	The comparison field is all hex FF.
	LOVALUE	The comparison field is all hex 00.

Term	Description
AND OR	When multiple or nested IF statements are used, this operand indicates whether the relationship between each IF is OR or AND. WHEN statements may be interspersed with IF statements.

OR Logic

The use of the OR connective is valid only when any number of IF...OR statements are followed by an IF statement. This is necessary to indicate the end of the OR logic. Any IF statements that follow it are not part of the preceding OR logic.

A found false condition causes the program to fall through to the next statement which must be an IF or WHEN. If the word OR is not included, a found true condition causes fall through to the next statement while a found false condition causes transfer to the next sentence.

AND Logic

If the word OR is not included, all multiple occurrences of IF statements must meet the conditions before the true statements are executed. (AND is inferred if OR is not present. AND may be specified for readability.)

If an IF statement fails, a fall through occurs to the next sentence — periods indicate the end of true condition processing (IF or WHEN) and identifies the next statement as the beginning of false processing (ELSE).

If FLDA equals A, B, or C, the true branch is to statement 50.

```
10 MOVE C'FALSE' TO PRT1
20 IF FLDA EQ C'A' OR
30      IF FLDA EQ C'B' OR          True
40      IF FLDA EQ C'C'
50      IF FLDB EQ C'1'
60      IF FLDC EQ C'$'
70    MOVE C'TRUE' TO PRT1.
80  PRINT
```

If FLDA equals X, the false branch is to statement 80.

```
10 MOVE C'FALSE' TO PRT1
20 IF FLDA EQ C'A' OR
30      IF FLDA EQ C'B' OR          False
40      IF FLDA EQ C'C'
50      IF FLDB EQ C'1'
60      IF FLDC EQ C'$'
70    MOVE C'TRUE' TO PRT1.
80  PRINT
```

Example

```
090 IF INF1 IS EQ C '_'              /* When all
100 IF INF20-21 IS EQ C'98'          /* Three IF
110 IF WST10 IS EQ C'?'              /* Statements are true,
120   MOVE INF1-80 TO PRT1           /* Statements
130   PRINT                          /* 120 through 140
140   GOTO 010.                      /* Are executed.
150 MOVE INF1-80 TO OFD1             /* When one or more are false,
160 *                                /* Control goes to statement 150
```

Arithmetic Comparison

When using IF statements involving arithmetic fields, comparison should be done in packed decimal format. This is the only way to compare numeric values with the signs of the two fields taken into account.

The following examples illustrate this point and also show the technique for testing for negative and/or positive fields:

```
IF WST11-15-P IS LT P'0'             /* Negative? (less than zero)

IF WST11-15-P IS GT P'0'             /* Positive? (greater than zero)
```

Numeric Comparison

When using IF statements in conjunction with the NUMERIC test, you must be aware of the contents of flddef. The following example illustrates this point and also shows the technique for testing numeric fields:

```
EQU     FLD1        WST1-4          C'123B'         /* X'F1F2F3C2'
EQU     FLD2        WST5-9
EQU     FLD2                                        /* Redefines FLD2
EQU     FL2A        WST5-8
EQU     FL2B        WST9            ZEROS

..

IF FLD1 NOT NUMERIC          /* A X'F1F2F3C2' is numeric (from unpacking?)
    PRINTHEX FLD1            /* Should not print
    GOTO xxx.
MOVE FLD1 TO FL2A            /* Move to work area
IF FLD2 IS NOT NUMERIC       /* A X'F1F2F3C2F0' is not numeric
    PRINTHEX FLD2.           /* Should print
```

The printout shows:

```
WST5-9  123B0
        FFFCF
    12320
       01..05
```

In the previous example, the first comparison for numeric finds that FLD1 is numeric. FLD1 may have been created from a COBOL program with a PICTURE S9(4) (or it could be a packed field unpacked into FLD1.) Since there is a valid sign in the low order position (X'C2'), FLD1 is a numeric field. After the MOVE statement, a numeric check of FLD2 shows that FLD2 is not numeric, as X'F1F2F3C2F0' is not considered numeric.

The previous example illustrates how to check for numeric fields where the low order position might have a sign. Note that a zone or sign of X'Dn' in the low order position is also valid (a negative number) just as a zone or sign of X'Cn' is considered positive.

C0 - C9 are positive numbers

D0 - D9 are negative numbers

IF (Compound)

Note: This is the compound IF statement. For the simple IF statement, see IF.

```
( compound condition )
```

A compound IF statement is formed by enclosing a compound condition with a preceding left parenthesis and a trailing right parenthesis.

A compound condition is the combination of a number of relation conditions joined by a number of enclosing parentheses and the logical operators AND and OR following the rules of logic.

A relation condition is composed of two operands joined by a relational operator. Relational operators include:

=	equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
<>	not equal

An operand is any QUIKJOB field definition or EQUate.

The compound IF statement follows the standard hierarchical rules for compound conditionals:

1. Conditions surrounding the word AND are evaluated first.
2. Conditions surrounding the word OR are evaluated last.
3. When there are several AND or OR logical operators, the AND conditions are evaluated first, as they appear in the statement, from left to right. Then the OR conditions are evaluated, also from left to right.
4. To override rules 1-3, use parentheses around the conditions to be evaluated first.

Example (Compounding):

```
EQU FILLER       WST0
EQU FLDA         (1)-P  P'1'
EQU FLDB         (2)-P  P'-1'
EQU FLDC         (2)-P  P'1'
EQU FLDD         (1)-P  P'1'
EQU FLDE         (1)-P  P'1'
EQU FLDF         (1)-P  P'1'
EQU FLDG         (1)-P  P'1'
    IF FLDA = FLDA
    IF (
     (
     ( FLDB TMP FLDC AND
       FLDC = FLDC  OR
       FLDD = FLDD ) OR
     ( FLDE = FLDE  OR
       FLDF = FLDF ) AND
     ( FLDE = FLDE  OR
       FLDF = FLDF ) AND
       FLDG = FLDG
       ) AND FLDC = FLDC
       )
       GOTO 999.
999    GOTO EOJ
```

Bit Testing

IF invokes the TEST UNDER MASK machine operation by specifying a field size of one byte and the following relational operators (see also the section Example 46 in Chapter 4):

- TMO = ones
- TMZ = zero
- TMNZ = not zero
- TMM = mixed

IF invokes the TEST UNDER MASK LOW machine operation by specifying a field size of two bytes and the following relational operators (see also the section Example 46 in Chapter 4):

- TMO = ones
- TMZ = zero
- TMNZ = not zero
- TMM = mixed and leftmost bit is zero
- TMP = mixed and leftmost bit is one

String Scanning

You can imbed the parameters for a WHEN statement in a compound IF statement. See TRNT FLDA WITH FLDB

```
    PRINTHEX FLDA
    PRINTHEX PTR1
    PRINTHEX VAL224-228
```

```
     WST1-20 BLUEJAYS EAT PEANUTS
             CDECDCEE4CCE4DCCDEEE
             23451182051307515432
            01..05...10...15...20
      PTR1-1 E
             C
             5
            01
VAL224-228
             00000
             10004
             01..05
```

WHEN for an explanation of the parameters.

Numeric Comparison

The IF statement can now do a numeric compare between unlike data types, such as character and packed. This may not produce the results that you desire. If you want to ensure that the comparisons are made prior to VISION:Report Release 16.0, use OPTION IFNUM=NO (normal default). See OPTION parameters.

ELSE and ENDIF

You can code false case logic by using an ELSE statement following the last statement in the true case and by using an ENDIF or a period following the last statement in the false case. The true case logic will branch to the statement following the ENDIF or period.

```
IF condition-1
      do-something-x
      [do-something-n]
ELSE
      do-something-y
      [do-something-n]
ENDIF
```

- An IF statement can be terminated by an ENDIF statement or a period (.).
- The use of an ENDIF statement and a period together is redundant.
- A period will terminate all preceding IF statements. An ENDIF statement will terminate only a single IF statement; IF/ENDIF statements must be paired.

Example:

- If WS-SUB equals zero, move 5 to WS-FLD1 and ABC to WS-FLD2
- If WS-SUB is not zero, WS-FLD1 would have a 7 and WS-FLD2 would have XYZ.

```
IF WS-SUB = ZERO
      MOVE C'5' TO WS-FLD1
      MOVE C'ABC' TO WS-FLD2
ELSE
      MOVE C'7' TO WS-FLD1
      MOVE C'XYZ' TO WS-FLD2.
```

Nested IF Syntax

```
IF (condition-1 [AND condition-2)] [OR condition-3]
   do-something-1
   [do-something-n]
   IF condition-4
         do-something-3
         [do-something-n]
   END-IF
ELSE
   do-something-5
   [do-something-n]
ENDIF.
```

Example:

```
If (WS-SUB equals zero and WS-X is greater than 3
      /* together, one condition above
OR WS-X equals zero)                /* or this condition
MOVE 5 to WS-FLD1                   /* end of first IF validation
     If WS-IDX is greater than a packed 7   /* second condition
            move 3 to WS-FLD3, move packed 4 to WS-FLD4
MOVE ABC to WS-FLD2
     /* gets executed regardless of any above IF conditions

If none of the above IF conditions are true, then execute the
following:
MOVE 7 to WS-FLD1
MOVE XYZ to WS-FLD2
```

```
IF ( ( WS-SUB = ZERO AND WS-X > 3 ) OR WS-X = 0)
    MOVE C'5' TO WS-FLD1
    IF WS-IDX > P'7'
    MOVE C'3' TO WS-FLD3
    MOVE P'4' TO WS-FLD4
END-IF
    MOVE C'ABC' TO WS-FLD2
ELSE
    MOVE C'7' TO WS-FLD1
    MOVE C'XYZ' TO WS-FLD2
ENDIF.
```

Example:

```
IF (WST1 = C'1' OR WST2 = C'1') AND
      IF WST3 = C'7'
```

```
        MOVE WST1-3 TO WST5-7
    ENDIF.
```

Note: The period after ENDIF terminates all preceding IF statements. We recommend that you use the following example.

Example:

```
IF (WST1 = C'1' OR WST2 = C'1') AND
      IF WST3 = C'7'
          MOVE WST1-3 TO WST5-7
      ENDIF
ENDIF
```

In the above example, the two ENDIF statements are correctly balanced and the last ENDIF terminates all preceding IF statements. We recommend that IF/ENDIF be paired, rather than having an imbalance and terminating with a period.

Example:

```
IF ( ( FLDA = FLDB  OR  FLDC = FLDD)  AND
       FLDE = FLDF )
    MOVE C'A=B OR C=D, E=F'   TO PRT1
ENDIF
```

Example:

```
IF  WST1 = C'1'
    MOVE WST1-3 TO WST5-7
ELSE
    MOVE WST5-6 TO WST7-8.
```

Nesting ELSE and ENDIF

IF allows nesting by the use of ELSE and ENDIF statements. Periods should not be used in combination with nesting statements.

Example (Nesting ELSE, ENDIF):

```
 IF FLDB = FLDB
    MOVE C'AAAAA' TO PRT20
 IF FLDB = FLDB
    MOVE C'BBBBB' TO PRT30
 ELSE
    MOVE C'CCCCC' TO PRT30
 ENDIF
 ELSE
    MOVE C'DDDDD' TO PRT40.

 IF FLDB = FLDB
    MOVE C'EEEEE' TO PRT40
 ELSE
    MOVE C'FFFFF' TO PRT50
 ENDIF
 ENDIF
 PRINT
```

IF allows the numerical comparison of unlike numerical formats, zoned, packed and binary, and unlike field sizes. If either of the fields is not numeric then a logical comparison is performed.

LIMITREADS

```
LIMITREADS nnnnnnn {INF|INA-INZ|DET}
```

This statement is declarative and therefore cannot be the subject of a transfer in control statement (for example, GOTO, PERFORM).

Term	Description
LIMITREADS	Limits the number of records read for a given file. This feature is useful for building test files, checking EOF logic with multiple input files, and testing reports to ensure control level totals occur correctly.
nnnnnnn	Maximum number of records VISION:Report reads from the specified input file.
INA-INZ, DET	File to be limited. INF is the default.

When the limit has been reached for the file or EOF occurs, VISION:Report places high-values (X'FF') in the input record area and an E in the VAL area to indicate EOF has occurred.

If the ATEND operand is specified on the GET statement or the ATEND statement was used, VISION:Report transfers to your end of file code.

If INF is the only input file and neither the ATEND operand on the GET nor the ATEND statement was used, VISION:Report proceeds to normal end of job processing.

If the specified file is closed and opened again, the LIMITREADS resets back to the original limit.

When using multiple input files, you are responsible for determining end of file and end of job. See GET for more information about end of file processing.

Examples

```
005 LIMITREADS 1000 INC
010 GET INC ATEND EOJ
020 MOVE INC1-100 TO OFA1
030 WRITE OFA
040 GOTO 010
```

In the previous example, the INC file is read and pulls the first 1000 records. When the 1001 record is read, VISION:Report proceeds to normal EOJ.

```
005 LIMITREADS 1000 INC
008 SAMPLE 10 INC
010 GET INC ATEND EOJ
020 MOVE INC1-100 TO OFA1
030 WRITE OFA
040 GOTO 010
```

This example is the same as the first, except for statement 008. A SAMPLE statement is used that causes, for example, the first, eleventh, record to be passed for processing. Only SAMPLE records passed determine the LIMITREADS count.

LINECOUNT

```
LINECOUNT {nnn|NONE}
```

Term	Description
LINECOUNT	Overrides the lines per page specified in the PRNTLCT option.
nnn	Enter a 1- to 3-digit value representing the number of lines to be printed on each page before automatic page ejection occurs.
NONE	If the word NONE is coded, the line count is to be ignored completely, giving you a page of infinite length.

LCT1-2-P contains the number of unused or available lines for use on the current page. In the following example, a page eject is forced if LCT1-2-P is less than 12.

Example

```
IF LCT1-2-P IS LT P'12'
     DOHEADERS.
       processing statements
```

LOAD

```
LOAD user-routine-name PTx
```

Term	Description
LOAD	Loads a user coded routine at execution time. Unlike the CALL statement, control is not passed to the routine, but instead a pointer is set to point to the first byte in the routine. The length of the routine is posted in VAL205-208 in binary format.
user- routine-name	Enter the name of the user routine to be loaded. The contents of this operand are used to retrieve the module from the VSE library.sublibrary or the appropriate MVS load library and must match exactly the name under which the routine was cataloged.
PTx	This operand must be one of the PTx index pointers (PTA, PTB, PTC, PTD, or PTR). It is set to point to the first byte of the routine.

The LOAD statement is normally used in those applications where a program was compiled and link edited. The program may also contain constants (DC) to be used as a table only with no executable code. The default VISION:Report translate table (see VISION:Report OPTION Keywords, TRLNAME) and default extended edit mask table (see VISION:Report OPTION Keywords, EDTNAME) are examples of non-executable routines.

Any number of references may be made to the same routine by either LOAD or CALL statements. However, only one actual loading of the routine from the library takes place.

Example

```
100 LOAD QUIKEMSK PTA                    /* MVS; if VSE, QUKBEMSK
```

This statement causes the module QUIKEMSK to be loaded, if not already loaded, places the address of the first byte of the routine in the PTA pointer, and places the binary length of the module in VAL205-208.

MOVCOND

```
MOVCOND flddef1 TO flddef2
```

Term	Description
MOVCOND	Condenses the 8-byte string of EBCDIC zeros and ones (F0/F1) in the source field to a single byte field with bit settings according to zero and one composition.
flddef1	Source field. Define an 8-byte field that must be comprised entirely of EBCDIC zeros (F0) and EBCDIC ones (F1).
TO	Required noise word.
flddef2	Result field. Define a 1-byte field into which the condensed source field is to be placed.

MOVCOND examines each byte of the source field from left to right and constructs a single byte, with bit 0 set to zero (if the leftmost source byte was zero) and with bit 0 set to 1 (if the leftmost source byte was one).

This evaluation and construction continues towards the right until all 8-byte/bit positions have been considered.

Examples

```
Source      Condensed
00000000    X'00'
10000000    X'80'
00000100    X'04'
```

MOVE

```
MOVE {flddef1|C'xxx'|P'nnn'|X'...'|ZERO|BLANK|SPACE|HIVALUE|LOVALUE} TO flddef2
[nC|nE|nN]
```

Term	Description
MOVE	Moves a data field or constant from any area to any other area. Editing and zero suppression are optionally available for numeric fields. See MOVE (Expanded Editing) for editing of non-quantitative fields. See MOVE (Variable Length) for moving variable length fields.
flddef1 C'xxx' P'nnn' X'...' ZERO BLANK SPACE HIVALUE LOVALUE	Describes the sending field. ■ A standard field definition or equated data name for data fields. ■ A character literal (C'xxx') up to 40 characters. ■ A packed literal (P'nnn') up to 15 digits in 8 bytes. ■ A hexadecimal literal (X'....') from 2 to 72 hexadecimal digits (36 bytes). ■ Constants (BLANK, SPACE, ZERO, LOVALUE, HIVALUE). (See General Rules, Rule L.)
flddef2	Describes the receiving field as a standard field definition or equated data name. When the sending field has a character format or when editing to PRT, you need only define the starting position of the receiving field.

More detailed coding rules for the sending and receiving fields are provided on the following pages.

Coding Rules	Description
nC, nE, nN	If moving to the PRT area (or any other area if EDITALL=YES) and editing and zero suppression is desired, code a 1- or 2-character field as follows: If a decimal point is to appear in a number, code a number in the range of 0 through 9 to indicate how many digits are to appear to the right of the decimal point. Code a C if commas are to be inserted as appropriate. Code an E if the European variation of commas / decimal are to appear. Code an N if zero suppression is not to be performed. If using an equated name, the edit code may be stated in the EQU statement. If moving to PRT and if this operand is blank and the equated name has appended to it an edit code, that edit code is used; otherwise this operand is used. (See EQU for more information.)

Examples

```
2C  Print 2 decimal positions and insert commas as appropriate.
1   Print 1 decimal point. No commas are shown.
C   No decimal point is shown but commas are inserted.
0N  No decimal point is shown, and zeros are not suppressed.
```

The option EDITALL increases the power in moving of fields with edit masks to areas other than the PRT area. If EDITALL=NO, then the edit masks or codes are only in effect if the target area is PRT. If EDITALL=YES, then the edit codes would affect all areas. Check with the person that installed VISION:Report to find out what option was installed as the default.

Warning: When the MOVE statement is used, the length of the target fields are ignored when moving character data. Target field length is honored only when moving a figurative constant such as SPACES or numeric data in packed format. Care must be taken to ensure that the length of the source field is correct, or else unpredictable results will occur when neighboring storage is overlaid, possibly resulting in abnormal terminations such as addressing, operation, or protection exceptions, depending upon the usage of the storage that is corrupted.

The allocation of storage and its placement for different releases of VISION:Report and the operating system will vary. Programs that were working under one release of VISION:Report and/or operating system can fail when either is upgraded. In the case of I/O areas, only the specified record size will be obtained. The following example will illustrate this, with several assumptions:

You issue a SET PTD to various locations within VISION:Report area OFA. The OFA area is 157 bytes long. The work field, WK-DATA2, is 100 bytes long. You issue the following instructions:

```
SET  PTD  OFA2
...
MOVE  WK-DATA2  TO  PTD65            /*  (1)
MOVE  WK-DATA2  TO  PTD116           /*  (2)
```

The first MOVE instruction would move data to OFA67-166, which is nine bytes longer than the length of OFA. Thus, storage following area OFA, which was 157 bytes, would be corrupted, and as mentioned above, the results would be completely unpredictable.

The second MOVE instruction would move data to OFA118-217, which is considerably more than what you would really want to move, with unpredictable results again.

You should correct the VISION:Report statements above to move only as much data as will not extend beyond the 157th byte of area OFA. One solution would be to use a variable length MOVE statement such as:

```
EQU   DATA-LENGTH                    WST300-301-B
...
MOVE  WK-DATA2  TO  PTD65     DATA-LENGTH
```

You would have to know or be able to compute the remaining length of the field, and then insert the value into DATA-LENGTH. See MOVE (Variable Length).

Another solution is to build the OFA record in a work area that has extra available storage at the end, and then move 157 bytes of the work area to OFA. With this method, you do not need to worry about corrupting storage not belonging to VISION:Report or computing the length of each move to the OFA record.

Data Conversion

The option MOVCVTX increases the power of moving literal X'....' to the target area, depending upon the target area data type. The default of MOVCVTX is NO. If MOVCVTX=YES, then the literal X'....' is converted according to the target area data type. Check with the person that installed VISION:Report to find out what option was installed as the default. Some examples follow (where xxxx represents whatever was in memory prior to the MOVE):

Sending Field	Receiving Field	MOVCVTX=NO Results	MOVCVTX=YES Results
X'012C'	WST1-2-P	X'012C'	X'300C'
X'012C'	WST1-2-B	X'012C'	X'012C'
X'012C'	WST1-4-P	X'012Cxxxx'	X'0000300C'
X'012C'	WST1-4-B	X'012Cxxxx'	X'0000012C'
X'012C'	WST1-4	X'012Cxxxx'	X'F0F3F0F0'

Detailed Coding Rules

Single quotation marks may be embedded in character literals by coding two consecutive quotation marks to indicate one data quotation mark. Thus C'"250"' generates as '250'.

Valid examples and results with single quotation marks in literal and the results:

```
MOVE C''''''    TO FLD1  /* Results: ''       (2 QUOTES)
MOVE C'''1'''   TO FLD1  /* Results: '1'      (QUOTE,1,QUOTE)
MOVE C''' '''   TO FLD1  /* Results: ' '      (QUOTE,BLANK,QUOTE)
MOVE C''''      TO FLD1  /* Results: '        (1 QUOTE)
MOVE C'''250''' TO FLD1  /* Results: '250'    (250 WITHIN QUOTES)
```

Negative values can be coded in packed literals by placing a hyphen followed by numeric digits. The same effect can be achieved by coding the character J for -1; K for -2; L for -3; and so forth through R for -9 (as the last digit).

One of the constants (ZERO, SPACE, BLANK, LOVALUE, HIVALUE) may be specified in the sending field. There is no length restriction on constants.

An EBCDIC to EBCDIC MOVE can be up to 2-gigabytes long and is executed in a single coded MOVE. Internally, it is executed from left to right using repeated MOVE instructions.

EBCDIC fields can be transformed to packed or binary by defining the receiving field with a -P or -B suffix.

EBCDIC fields (up to 19 bytes long) may be edited to the print line (PRT area) by including editing and zero suppression specifications.

A packed field (up to 10 bytes) can be transformed to EBCDIC or binary or could be edited to the print line (PRT area) by including editing and zero suppression specifications.

Binary fields may be transformed to EBCDIC or packed and may also be edited to the print line (PRT area) by including editing and zero suppression specifications. Binary sending fields may be 1-to 8-bytes long.

Options

BLANK WHEN ZERO (BWZ) is a VISION:Report option that suppresses printing zero-filled counters.

- If OPTION BWZ=NO is in effect, an ACCUM statement specifying 2C prints as .00.
- If OPTION BWZ=YES is in effect, an ACCUM statement specifying 2C prints as a blank field.

You also have the option of designating whether or not negative amounts in counters print with a negative sign (-) or credit character (CR).

- If OPTION CRSIGN=NO causes a negative sign to print, indicating a negative counter.
- If OPTION CRSIGN=YES causes CR to print, indicating a negative counter count. The use of CR also requires one additional position to be used in the print line.

MOVE Examples Using Various Editing Codes and Options

OPERAND 3	SOURCE DATA	BWZ=NO, CRSIGN=NO EDITED RESULT	BWZ=YES, CRSIGN=YES EDITED RESULT
	000012345	000012345	000012345
0	000012345	12345	12345
0C	000012345	12,345	12,345
0C	-000012345	12,345-	12,345CR
0E	000012345	12.345	12.345
0N	000012345	000012345	000012345
0N	000000000	000000000	000000000
1	000123456	12345.6	12345.6
1C	000123456	12,345.6	12,345.6
1C	000000000	.0	
2	001020345	10203.45	10203.45
2C	001020345	10,203.45	10,203.45
2C	-001020345	10,203.45-	10,203.45CR
2C	000000000	.00	
2E	001122334	11.223,34	11.223,34
2E	-001122334	11.223,34-	11.223,34CR
2E	000000000	,00	
2N	001122334	011223.34	0011223.34
3	009876543	9876.543	9876.543
3C	009876543	9,876.543	9,876.543
3C	-009876543	9,876.543-	9,876.543CR
3C	000000000	.000	

Samples and Rules

MOVE INF5-7 TO OFA10-12	Move input file positions 5-7 to output A positions 10-12 (length of target is ignored).
MOVE WST1-5-P TO PUN1-9	Unpack to EBCDIC format. F zone forced on positive fields.
MOVE INF3-6 TO OFA1-3-P	Pack EBCDIC. Limit of 19 digits source field length, 10-byte packed destination field.
MOVE INF14-20 TO PRT25	EBCDIC to EBCDIC. No TO element required for PRT destination.
*MOVE INF15-20 TO PRT25 2C MOVE INF15-20 TO PRT25 2E	EBCDIC to EBCDIC, but final operand identifies a numeric source field that is to be zero-suppressed in PRT and have a decimal point inserted to the left of the 2 low order print positions.

*MOVE WST1-3-P TO PRT30 3C MOVE WST1-3-P TO PRT30 3E	Packed to EBCDIC in PRT with zero suppression and a decimal point to be inserted to the left of the 3 low order print positions.
MOVE C'ABCDEFGHIJKLMNOPQRSTUVWXYZ12345678912345' TO OFA1-40	Limit of 40 characters (length of target is ignored).
MOVE P'15' TO OFX1-6-P	Limit of 19 digits in literal and 10 bytes in destination.
MOVE X'45E0' TO OFB1-2	Limit of 36 bytes in destination.
MOVE SPACES TO OFA1-300	No limit on length, BLANKS may also be used.
MOVE INF5-10 TO WST1	The sending field (INF5-10) determines the length of data movement where no data conversion and/or editing is required. Six positions are moved.
MOVE ZEROES TO WST1-5-P	Limit of 10 bytes when moving to a packed field. This example would have packed zeros due to the P specification on the destination field.
MOVE HIVALUES TO OFZ1-10000	No limit on length. Moves X'FF'.
MOVE LOVALUES TO OFB20-20000	No limit on length. Moves X'00'.
MOVE C'85' TO OFA1-2-B	The B destination specification causes the literal to be converted to the fixed point binary representation of the C' ' value; in this case OFA1-2-B would be set to X'0055', the binary representation of decimal 85. The destination field may be up to 8 bytes, with no boundary alignment required.
MOVE INF15-20-P TO PRT25 2C	Packed to print with editing. The C after the 2 is optional and causes commas to be inserted at the appropriate places in the result.

* - The C operand in connection with the decimal operand when moving to the printer is optional. When included, this causes commas to be inserted in the appropriate places in the results. An E causes European punctuation to be generated.

MOVE INF1-5-B TO WST1-6-P	**The B specification causes the 5-byte binary field to be converted to packed decimal. Binary fields of 1-8 bytes may be converted to packed decimal.
MOVE INF1-2-B TO OFA78-83	**The B specification causes the 2-byte binary field to be converted to zoned decimal. Binary fields of 1-8 bytes may be converted to zoned decimal.
MOVE WST7-10-B TO PRT11 0C	Move binary fields of 1-8 bytes long to print with editing.

** - It is your responsibility to specify a large enough receiving field for conversion. Excess is truncated and no warning is given.

Move Allowable Forms

From	To	Allowable
EBCDIC	EBCDIC	Yes
EBCDIC	Packed	Yes
EBCDIC	Binary	Yes
EBCDIC	Print No Edit	Yes
EBCDIC	Print Edit	Yes
Packed	Packed	Yes
Packed	EBCDIC	Yes
Packed	Print Edit	Yes
Literal C'_'	EBCDIC	Yes
Literal C'nnn'	Packed	Yes
Literal C'nnn'	Print No Edit	Yes
Literal C'nnn'	Print Edit	Yes
Literal P'n'	Packed	Yes
Literal P'nnn'	EBCDIC	Yes
Literal P'nnn'	Print No Edit	Yes
Literal P'nnn'	Print Edit	Yes
Literal X'_'	EBCDIC	Yes (Limit of 36 bytes)
ZERO	Packed	Yes (Limit of 10 bytes)
SPACE,BLANK		
ZERO,HIVALUE		
LOVALUE	EBCDIC	Yes (No Length Limit)
Binary	EBCDIC Zoned Decimal	Yes
Binary	Packed Decimal	Yes
Binary	Print Edit	Yes

Move Print Position Requirements

The number of decimal positions need to be accounted for in the total number of print positions.

- C inserts commas as appropriate.
- E inserts a period before every third integer position, and a comma in the decimal position.
- N suppresses leading zeros.

The area required in the PRT area is:

- One byte for each data byte.
- One byte for a minus sign if CRSIGN=NO. Two bytes for the literal CR if CRSIGN=YES.
- One byte for decimal point if decimals are specified.
- One byte for every three integers if C or E is specified.

The use of option on CRSIGN=YES requires one additional print position to the number indicated above.

Binary-Move Print Position Requirements

Binary fields require print positions to accommodate the maximum digits possible.

Data Length Binary	Number of Digits
1	3
2	5
3	7
4	10
5	12
6	15
7	17
8	19

MOVE (Variable Length)

```
MOVE {flddef1|C'xxx'|P'nnn'|X'...'|ZERO|BLANK|SPACE|HIVALUE|LOVALUE} TO flddef2
{flddef3|X'...'}
```

Term	Description
MOVE	Moves a data field or constant from any area to any other area with a variable length on the move. Editing and zero suppression are not applicable with a variable length move.
flddef1 C'xxx' P'nnn' X'...' ZERO BLANK SPACE HIVALUE LOVALUE	Describes the sending field. ■ A standard field definition or equated data name for data fields. ■ A character literal (C'xxx') up to 40 characters. ■ A packed literal (P'nnn') up to 19 digits in 10 bytes. ■ A hexadecimal literal (X'....') from 2 to 72 hexadecimal digits (36 bytes). ■ Constants (BLANK, SPACE, ZERO, LOVALUE, HIVALUE). (See General Rules, Rule L.)
TO	Required noise word.
flddef2	Describes the receiving field as a standard field definition or equated data name. You need only define the starting position of the receiving field. Sending and receiving fields must be in the same format. However, hexadecimal fields may be moved to character fields.
flddef3 X'...'	Describes this operand as a standard field definition, an equated data name, or hexadecimal literal. This flddef must be a 2-byte or 4-byte binary field that contains the desired length of the number of bytes to be moved.

If an equated data name is used, the name must be at least three characters long, or you can get unpredictable results.

Note: No data format transformation occurs from the sending to receiving operands, unless the sending field is a hexadecimal literal (X'...') and the option is MOVCVTX=YES. It is important to maintain data integrity. Moves made from one field to another without regard to formats could result in damaged data (for example, dropping the sign portion of a packed field) that can cause VISION:Report errors.

Example

```
010 GET INF ATEND EOJ                          /* Get fixed length record
020 WHEN INF1-80 INCLUDES X'FF'                /* Find end of fixed record
030 MOVE INF1 TO OFA5 VAL225-228-B             /* Move fixed RCD to variable OFA
040 MOVE VAL227-228-B TO OFA1-2-B              /* Move RCD length to OFA1
050 WRITE OFA.                                 /* Write OFA
060 GO TO 010                                  /* Go back and get next record
```

Statement 030 shows a variable length move from INF1 to OFA5, depending upon the contents of VAL225-228-B.

MOVE (Expanded Editing)

```
MOVE {flddef1|C'xxx'|P'nnn'|X'...'} TO flddef2 mask-code
```

Term	Description
MOVE	Moves a data field from one location to another and edits the data into a prescribed pattern. This function is used to move dates, times, social security numbers, and telephone numbers to a target area, inserting hyphens, slashes, and periods. An edit mask pattern determines the placement of source data and punctuation insertion. This MOVE with expanded editing is for moving non-quantitative fields.
flddef1 C'xxx' P'nnn' X'...'	Source data to be moved and edited. This field can be: ■ A character literal up to 40 bytes long. ■ An EBCDIC field up to 63 bytes long. ■ A binary field from 1 to 8 bytes long. ■ A packed field with a maximum of 10 bytes long. Also, the expanded MOVE statement automatically handles a packed unsigned (no sign) field and treats it as a positive field.
TO	Required noise word.
flddef2	Area/field for the edited result. Usually, this is PRT, but any VISION:Report area is valid. The length of the result field equals the length of the mask-code.
mask-code	An alphabetic character from the table that follows, or a character assigned for a special purpose or use at your installation. The letters C, E, or N cannot be used for a mask code since these are used by VISION:Report to designate decimal and comma punctuation.

If an equated data name is used, the name must be at least three characters long or you will get unpredictable results.

The option EDITALL increases the power for moving fields with edit masks to areas other than the PRT area. If EDITALL=NO, then the edit masks or codes are only in effect if the target area is PRT. If EDITALL=YES, then the edit codes would affect all areas. Check with the person that installed VISION:Report to find out what option was installed as the default.

Examples

Sending Field	Receiving Field	EDITALL=NO Results	EDITALL=YES Results
123456789	PRT1 S	'123-45-6789'	'123-45-6789' (11 bytes)
123456789	OFA1 S	Not accepted	'123-45-6789' (11 bytes)

Edit Mask Patterns

Nine commonly used edit mask patterns are distributed with the VISION:Report system. The systems programmer or person in charge of software systems has the necessary information to add or change edit mask patterns as required for an installation's needs.

An installation may have up to 23 different edit masks in any one mask table. Multiple tables are allowed, but only one can be used for any execution of VISION:Report. See EDTNAME OPTION in the OPTION statement.

The QJEDIT macro is furnished on the VISION:Report distribution tape for this purpose. Mask patterns consist of data selector characters, and punctuation or edit fill characters.

Data Selector Characters

The following characters are used to indicate positions in the output field that contain characters from the input or source field.

Character	Description
9	This character indicates that the output field position is a numeric character. If the FORCE=YES (see Edit Mask Attributes) indicator has been set, a 9-character is required in the low order selector position of the mask.
Z	This character has the same meaning and rules as the 9 except leading zeros are suppressed (spaces are substituted) when JUSTIFY=RIGHT (see Edit Mask Attributes).
X	This character is used to indicate that the output field position contains an alphanumeric character.

Edit Fill Character

The character B when used in the mask causes a blank or space to be placed in the output field position.

Punctuation Characters

Any character (other than the 9, Z, X, and B) appearing in the masks is unchanged in the output field position it occupies.

Edit Mask Attributes

The following two operand parameters are optional. Default values are indicated when they are omitted.

Operand	Description	
FORCE	YES	VISION:Report forces the sign of the field to be positive, if the lower rightmost selector position has a 9 or a Z in it.
	NO	This is the default value. No sign checking or forcing is performed.
JUSTIFY	This operand parameter may contain one, two, or three sublist items. Each sublist item is a specification of one of two possible parameters. If more than one item is entered, the list must be enclosed in parentheses. The first item in the list must be RIGHT or LEFT.	
	RIGHT	This is the default value for this pair of parameters. The data field is edited starting at the right side of the pattern. The edit proceeds from right to left.
	LEFT	The data field is edited starting at the left side of the pattern. The edit proceeds from left to right.

One of the next item pairs may appear in either the second or third position of the sublist.

Term	Description
TRUNC	Any data in excess of the data selector positions in the mask pattern is truncated (dropped) without any checking or warning.
NOTRUNC	This is the default value of this item pair. Data in excess of the mask pattern prompts VISION:Report to issue an error message. VISION:Report allows one data position extra to appear in the packed or binary field than in the edited output.
PROPGAT	If the source field contains fewer characters than expected for that edit mask pattern, VISION:Report propagates spaces when JUSTIFY=LEFT, or propagates zeroes when JUSTIFY=RIGHT, to complete the mask pattern in the output.
NOPROPGAT	This is the default value for this item pair. VISION:Report checks for a MOVE that has fewer characters of input than what is expected for the output edited field. VISION:Report allows one less data position if the data field is binary or packed when this parameter is specified.

Source Field Data Formats

Term	Description
EBCDIC	The data field is edited into the mask pattern 1 byte at a time. No packing of the data is done.
Packed	The data field is unpacked into a work area. It is edited into the mask pattern 1 byte at a time. The packed data field is tested to see if it contains a valid sign in the low order half byte; one is provided if necessary.

Term	Description
Binary	The binary data is converted to EBCDIC format into an internal work area. The data is then edited into the mask pattern, 1 byte at a time. The following binary field lengths are assumed to contain the number of digits indicated.

Field Size	Number of Digits
1	3
2	5
3	7
4	10
5	12
6	15
7	17
8	19

VISION:Report Supplied Patterns and Attributes

Code	Pattern	Justify	Truncate	Propagate	Force
A	99/99	Right	Yes	Yes	Yes
B	99.99	Right	Yes	Yes	Yes
D	99/99/99	Right	Yes	Yes	Yes
F	99-99-99	Right	Yes	Yes	Yes
G	99.99.99	Right	Yes	Yes	Yes
H	ZZ9	Right	Yes	Yes	Yes
L	999-9999	Right	Yes	Yes	Yes
S	999-99-9999	Right	Yes	Yes	Yes
T	999-999-9999	Right	Yes	Yes	Yes
U	(999)B999-9999	Right	Yes	Yes	Yes
Y	99/99/9999	Right	Yes	Yes	Yes

Examples

Statement	Source	Result
MOVE INF1-6 TO PRT11 D	060196	06/01/96
MOVE INF1-6-P TO PRT11 T	05134359514	513-435-9514
MOVE INF26-31 TO PRT11 G	122536	12.25.36
MOVE INF1-4-P TO PRT24 A	0001296	12/96
MOVE INF1-4 TO PRT60 D	1296	00/12/96

In the following example, an edit mask with a code of W is developed to edit an account number.

```
MASK        99-9999-999
JUSTIFY     RIGHT
TRUNC       NO
PROPAGAT    NO
FORCE       NO
```

Statement	Source	Result
MOVE ACCOUNT-NO TO PRT10 W	019135731	01-9135-731

In the following example, an edit mask with a code of P is developed to edit a part number.

```
MASK        Z9-999BXXXBXX-9
JUSTIFY     LEFT
TRUNC       NO
PROPAGAT    NO
FORCE       NO
```

Statement	Source	Result
MOVE PART-NO TO PRT1 P	01347ABCGN3	1-347 ABC GN-3

In the following example, an edit mask with a code of R is developed to print names with initials.

```
MASK        X.BX.BXXXXXXXXXXXXXXXXX
JUSTIFY     LEFT
TRUNC       YES
PROPAGAT    YES
FORCE       NO
```

Statement	Source	Result
MOVE DB-NAME TO PRT11 R	JASMITHE	J. A. SMITHE
	CDJONES	C. D. JONES

MOVEXPD

```
MOVEXPD flddef1 TO flddef2
```

Term	Description
MOVEXPD	Expands the 8 bits at the 1-byte source field into an 8-byte EBCDIC string of zeros and ones.
flddef1	Defines a 1-byte field whose bits are to be expanded to bytes.
TO	Required noise word.
flddef2	Defines an 8-byte field into which the expanded representation of the source field is placed.

MOVEXPD examines each bit (left to right) of the source field. It translates the value of each bit to a printable character (0 or 1) and replaces it at the corresponding byte of the receiving field.

Example

```
Source          Expanded
X'C1'           X'F1F1F0F0F0F0F0F1'
```

MOVNUM

```
MOVNUM {flddef1|X'...'} TO flddef2
```

Term	Description
MOVNUM	Moves up to 2 gigabytes of binary numeric bits (4-7) from the sending data field to the receiving data field.
flddef1 X'...'	Sending field. Defines the numeric bits (4-7) field to be moved to receiving field.
TO	Required noise word.
flddef2	Receiving field. The field that contains the numeric bits (4-7) after the move.

Example

```
010 MOVNUM INF1-50 TO OFA1-50
```

The following statement forces a negative sign on a packed decimal field:

```
010 MOVNUM X'0D' TO INF4
```

To properly initialize a working storage area prior to a MOVNUM command, the area should be initialized with hi-values (X'FF') or zeros (X'F0'). This will then obtain positive results.

Example

```
MOVE HIVALUES TO WST1-8
MOVNUM INF1-8 TO WST1-8
```

MOVZON

```
MOVZON {flddef1|X'...'} TO flddef2
```

Term	Description
MOVZON	Moves up to 2 gigabytes of binary zone bits (0-3) from the sending data field to the receiving data field.
flddef1 X'...'	Sending field. Defines the zone bits (0-3) field to be moved to the receiving field.
TO	Required noise word.
flddef2	Receiving field. The field that contains the zone bits (0-3) after the move.

The following statement forces a negative sign over the units position of an EBCDIC numeric field.

```
010 MOVZON X'D0' TO INF32
```

This sequence moves the sign from the high order position of a field (position 24) to the low order position of the field (position 32) and then clears the sign from the high order position.

```
010  MOVZON  INF24  TO INF32
020  MOVZON  X'F0'  TO INF24
```

Example

```
010  MOVZON  INF1-50 TO OFA1-50
```

MSHIFT

```
MSHIFT {flddef1|C'nnn'} TO flddef2 n [LEFT|RIGHT] [R]
```

Term	Description
MSHIFT	Moves and shifts the data for scaling of numeric fields.
flddef1 C'nnn'	Source field that contains the numeric data to be moved and shifted, coded in the standard area/definition format. Data may be in EBCDIC format (maximum of 19 digits), packed format (maximum of 10 bytes), or binary format (maximum of 8 bytes). You may code a character literal with a maximum of 11 digits in the constant.
TO	Required noise word.
flddef2	Destination field, which has the same format and length limitations as the source field. The actual shifting is done in a work area, allowing sending and receiving locations to overlap or be the same.
n	Number of digits to be shifted in the data field, with a value ranging from 0 through 9.
LEFT RIGHT	Directional shift of the data. LEFT adds zeros to the low order positions in the field. This, in effect, is a multiplication by a factor of 10. RIGHT truncates the low order positions of the field. This, in effect, is a division by a factor of 10. To determine the 10s factor, merely use n (number of digits) as the exponent. For example, if n=3, then the factor is 10 x 10 x 10 or 1000.
R	Rounds the data field contents. This only affects an MSHIFT with a RIGHT direction shift.

The sign of the sending field is carried to the receiving field.

This statement is implemented for scaling use with VISION:Forms, but is available for VISION:Report users as needed.

Examples

```
                                          Source      Destination
MSHIFT INF11-12-P TO WST1-4-P 2 LEFT      123         0012300
MSHIFT CTA4-8-P TO WST21-26 4 RIGHT       000072044   000007
```

MULT

```
MULT flddef1 nD BY {flddef2|C'xxx'|P'nnn'|X'...'} nD GIVING flddef3 nD[R]
```

Term	Description
MULT	Multiplies the contents of flddef1 by the contents of flddef2, a literal, or a constant, and stores the product in the field defined by flddef3. Decimal alignment specifications are required for all operands. Fields and literals in any data format can be multiplied by fields in any other data format. Automatic data conversion is performed in all cases.
flddef1	Multiplicand field in standard format. Data can be in EBCDIC format (maximum of 19 digits), packed format (maximum of 10 bytes), or binary format (maximum of 8 bytes).
nD	Number of decimal positions in multiplicand field (for example, if the multiplicand field has 2 decimals, code 2D).
BY	Required noise word.
flddef2 C'xxx' P'nnn' X'...'	Multiplier field in standard format with the same length limitations as the multiplicand field. You can also code a constant or a literal as the multiplier with a maximum of 11 digits in the constant of literal.
nD	Number of decimal positions in the multiplier field
GIVING	Required noise word.
flddef3	Product field in standard format with the same length limitations as the multiplicand field.
nD	Number of decimal positions in the product field.
R	Rounds the product. This keyword must immediately follow the nD keyword (such as 2DR). There cannot be a space in between the two keywords.

All multiplication and division is done internally in packed decimal. Fields to be multiplied may be up to 10 bytes for packed decimal, 19 bytes for EBCDIC, and 8 bytes for binary. The product may be up to 10 bytes for packed decimal, 19 bytes for EBCDIC, and 8 bytes for binary (10 digits with a maximum value of 2,147,483,647 and a minimum value of -2,147,483,648). An attempt to compute a binary product greater than 2,147,483,647 or less than -2,147,483,648 results in a FIXED POINT DIVIDE exception.

OPEN

```
OPEN {INF|INA-INZ|DET|OFA-OFZ} [password] [RESET]
```

VISION:Report automatically opens all files that do not have an OPEN statement.

Term	Description
OPEN	Defers the opening of an input or output file until you need it, and allows a password to be supplied for protected VSAM files. When used after a CLOSE statement allows an input or output file to be reopened for additional processing.
INA-INZ DET OFA-OFZ	File name can be any VISION:Report input or output file name (INF, OFA, INX, as examples) If this operand is left blank, INF is assumed.
password	If this operand is specified, the file must be a VSAM file and the password must be 8 bytes, padded with spaces if necessary. The password can be a field definition, a character literal, or a hexadecimal literal.
RESET	Overwrites the existing contents of an output VSAM file defined with REUSE.

The checking by VISION:Report to ensure that a reference to a file area is not greater than the record size (ERR208) is delayed until the OPEN statement is executed.

Example

Valid:

```
005 MOVE ZEROES TO WST1-2                /* Clear file flags
010 GET DET                              /* File opened automatically
020 IF DET1-3 IS EQ C'INC'               /* Want to open INC?
030   MOVE C'1' to WST1
040     OPEN INC                         /* Open
050   GOTO 100.
060 IF DET1-3 IS EQ C'IND'               /* Want to open IND?
070   MOVE C'1' to WST2
080     OPEN IND                         /* Open
085   GOTO 100.
095 OPEN INF                             /* Open INF. (INF did not need to
100   Processing statements              /*  be specified. It opens by default.)
```

```
010 GET INC ATEND 100                    /* File automatically open
      Processing statements
100 CLOSE INC
110 OPEN INC
120 GET INC ATEND EOJ
      Processing statements
010 GET INF ATEND 100
      Processing statements
100 OPEN DET
110 GET DET ATEND EOJ
      Processing statements
```

Invalid:

```
010 GET DET ATEND 100
      Processing statements
100 OPEN DET
110 GET DET ATEND EOJ
      Processing statements
```

A CLOSE statement is required at statement 100 followed by an OPEN statement.

Note: If a file is OPENed, it should also be CLOSEd.

OPTION

```
OPTION keyword={nnnn|YES|NO|name} [,keyword={nnnn|YES|NO|name}] ...
```

Term	Description
OPTION	Overrides or changes various control functions for one execution of VISION:Report. If used, VISION:Report OPTION statements must be the first statements.
keyword=	Defines which option to override. See Summary of Job Options and the section following for detailed descriptions of each keyword's function.

VISION:Report options are made available on three levels. They are provided for all systems as follows:

Option	Description
Default	These are distributed with the VISION:Report system. Their respective defaults and values are noted on the following pages.
Installation	One or more of the OPTION defaults can be changed permanently at a VISION:Report installation. A QJOPTION macro is supplied on the distribution tape to incorporate and catalog the options new defaults permanently. The systems programmer or person in charge of software systems at each installation has the necessary information and instructions available to perform this function.
User	One or more of the options may be changed or overridden at VISION:Report execution time. The format of the OPTION command is the word OPTION starting in any column (if starting in other than column 1, all preceding columns must be spaces), followed by at least one space, followed by operands that may extend through column 72. Columns 73-80 of the OPTION command are ignored by VISION:Report. The operands must be a continuous character string separated by commas with no embedded spaces; all columns past the first space are considered comments. The operand string must end with a complete operand without a trailing comma. If there are more operands than fit on a statement, simply code additional OPTION statements. There is no continuation indicator.

Option	Description
Obsolete	The following OPTION statements are no longer required, but remain in VISION:Report for compatibility: ■ #EQU ■ CALLCT ■ CALLSZ ■ CDIOUR ■ GENSIZE ■ LITSIZE ■ SORTMAX ■ SORTMIN ■ STMTS They may still be coded within the VISION:Report program, but the coding has no effect and is ignored.

Example

```
OPTION LIST=NO
    OPTION SEQCHK=YES,CLRVIP=NO
    OPTION MOVCVTX=YES,SRTADJ=YES
```

If the same option is expressed more than once, the last specification is used. PRTDD, LIST, PRTSIZE, and STMTLCT can be used only in the first OPTION statement. Use the following table to keep a record of the default options at your installation.

Summary of Job Options

Keyword	VSE Default	MVS Default	Installation Default
#EQU	N/A	N/A	
BWZ	No	No	
CALLCT	N/A	N/A	
CALLSZ	N/A	N/A	
CDIOUR	N/A	N/A	

Keyword	VSE Default	MVS Default	Installation Default
CFLEOPT	No	No	
CLRVIP	No	No	
CLRVOP	No	No	
CRSIGN	No	No	
DBIRTN	No	N/A	
DELUPGM	N/A	Yes	
DETDD	N/A	SYSDET	
DUMPALL	No	N/A	
EDIT	No	No	
EDITALL	No	No	
EDTNAME	QUKBEMSK	QUIKEMSK	
EUROPTN	No	No	
EXPMLOG	No	No	
EXPMLST	Yes	Yes	
GENSIZE	N/A	N/A	
HDRDOTS	Yes	Yes	
HOSTRTN	No	N/A	
IFNUM	No	No	
INADD	N/A	SYSINA	
INBDD	N/A	SYSINB	
INCDD	N/A	SYSINC	
INDD	N/A	SYSIN	
INDDD	N/A	SYSIND	
INEDD	N/A	SYSINE	
INFDD	N/A	SYSUT1	
INGDD thru INZDD	N/A	SYSING thru SYSINZ	
LIST	Yes	Yes	
LISTABL	No	No	
LISTOPT	No	No	

Keyword	VSE Default	MVS Default	Installation Default
LITSIZE	N/A	N/A	
MBUFFER	Yes	N/A	
MOVCVTX	No	No	
MSGROLL	N/A	Yes	
OFADD	N/A	SYSUT2	
OFBDD	N/A	SYSUT3	
OFCDD	N/A	SYSUT4	
OFDDD	N/A	SYSUT5	
OFEDD thru OFZDD	N/A	SYSOFE thru SYSOFZ	
OVLY	N/A	No	
PARMEXE	No	N/A	
PARMFLD	N/A	Yes	
PFLEOPT	No	No	
PRNTLCT	54	54	
PRODCOD	-	-	
PRTDD	N/A	SYSPRINT	
PRTSIZE	133	133	
PUNDD	N/A	SYSPUNCH	
PUNSIZE	N/A	81	
QJMDUMP	Yes	Yes	
RESVMEM	00	N/A	
RPTDD	N/A	SYSPRINT	
RPTSPCE	0	0	
RPTSYS	000	N/A	
SAVAREA	256	256	
SEQCHK	Yes	Yes	
SORTABL	Yes	Yes	
SORTMAX	N/A	N/A	
SORTMIN	N/A	N/A	

Keyword	VSE Default	MVS Default	Installation Default
SORTPRT	No	N / A	
SORTRTE	BOTH	N / A	
SORTSIZ	40	N / A	
SORTSYS	001	N / A	
SORTWRK	1	N / A	
SPIE	N / A	Yes	
SQLA1	0	0	
SQLA2	0	0	
SQLA3	0	0	
SQLA4	0	0	
SQLA5	0	0	
SQLPLNM	********	********	
SQLSYSN	****	****	
SQLVER	00.00	00.00	
SRTADJ	No	No	
SRTERCD	N / A	150000	
SRTMSG	N / A	??	
SRTPGM	N / A	SORT	
SRTSIZE	N / A	0	
SRTWKN	N / A	No	
STMTEND	80	80	
STMTIN	SYSIPT	N / A	
STMTLCT	50	50	
STMTS	N / A	N / A	
STXITPC	Yes	N / A	
SUBSPIE	N / A	Yes	
TRACECT	10	10	
TRLNAME	QUKBTRN	QUIKTRNT	
UABNDMP	No	No	

Keyword	VSE Default	MVS Default	Installation Default
UEXIT1	No	No	
U331DMP	Yes	Yes	
U333ABE	No	No	
U334DMP	N/A	No	
U335DMP	Yes	Yes	
U336DMP	Yes	Yes	
U339DMP	No	No	
VLABEND	N/A	Yes	
WSTSIZE	1000	1000	
ZEROPRT	No	No	

VISION:Report OPTION Keywords

The following list contains all of the valid VISION:Report OPTION keywords. All are applied to all systems, except where VSE ONLY or MVS ONLY is noted. The underlined operand is the VISION:Report default condition or value that is in effect at execution time.

Note: The PRTSIZE, PRTDD, LIST, or STMTLCT option must be on the first OPTION statement.

To obtain a printout of the current installation default options, code `OPTION LISTOPT=YES`. All options except `PRODCOD` are shown.

```
BWZ=NO
    YES
```

Blank-When-Zero. When a source field is zeros on a MOVE to PRT with at least 1 decimal place, should VISION:Report leave the area blank (instead of .00)? This is a run-time option and applies to all MOVE and ACCUM statements with decimals involved.

```
CFLEOPT=NO
        YES
```

File counts on system console at end of job?

```
CLRVIP=NO
       YES
```

Clear the entire input area to spaces before reading a variable length record?

```
CLRVOP=NO
       YES
```

Clear the entire output area to spaces after writing a variable length record?

```
CRSIGN=NO
       YES
```

Should VISION:Report indicate negative fields moved to PRT with CR instead of the usual negative sign (-)? This is a run-time option and applies to all MOVE and ACCUM statements with editing involved.

```
DBIRTN=NO
       YES                              (VSE ONLY)
```

Should VISION:Report return to the database interface on abnormal conditions or termination? This is only effective when used in conjunction with one of the VISION:Report interfaces (QUIKDLI, QUIKIDMS, as examples).

```
DELUPGM=YES
        NO
```

Delete user programs referenced in CALL/LOAD statements at EOJ? Modules performing I/O to data set names without CLOSE should specify NO.

```
DETDD=SYSDET
      ddname                            (MVS ONLY)
```

The ddname for the VISION:Report DET file.

```
DUMPALL=NO
        YES                             (VSE ONLY)
```

Should VISION:Report PDUMP the critical portions of the supervisor, as well as the partition GETVIS area in VSE systems when a memory dump is to be taken?

```
Edit=No
     Yes
```

Should VISION:Report commands be compiled, but not executed? If EDIT=YES, the commands are syntax checked and, if a report is specified, a small proof copy is produced.

```
EDITALL=NO
        YES
```

If NO, edit masks or codes are in effect only if target area is PRT. If YES, edit codes are in effect for all areas.

```
EDTNAME=QUIKEMSK            (MVS)
        username

EDTNAME=QUKBEMSK            (VSE)
        username
```

Load module/phase name of the expanded MOVE edit mask pattern table. QUIKEMSK (MVS) or QUKBEMSK (VSE) is the name of the module supplied with the system and contains mask patterns as explained in the expanded MOVE statement.

```
EUROPTN=NO
       YES
```

Are European variation of commas / decimals to be used exclusively? Applicable to MOVE TO PRT and REPORT fields with print specifications.

```
EXPMLOG=NO
       YES
```

Display expiration message on the system console? When VISION:Report product code is within 45 days of expiring, a warning message specifying the number of days left is displayed each time VISION:Report is run.

Do not select EXPMLOG=NO and EXPMLST=NO as the standard options, as you will not know when VISION:Report will expire.

```
EXPMLST=YES
       NO
```

Print expiration message (SYSPRINT if MVS, SYSLST if VSE)? When VISION:Report product code is within 45 days of expiring, a warning message specifying the number of days left is displayed each time VISION:Report is run. The EXPMLST option allows you to specify if you want the expiration message to appear on the printer.

Do not select EXPMLOG=NO and EXPMLST=NO as the standard options, as you will not know when VISION:Report will expire.

```
HDRDOTS=YES
       NO
```

Should generated field headings use periods (.) as the fill character for header / data field alignment? Blanks are used when NO is in effect.

```
HOSTRTN=NO
       YES                          (VSE ONLY)
```

Should numeric fields with unlike data types be compared numerically?

```
IFNUM=NO
      YES
```

Should VISION:Report return control to the host caller (by Register 14) at end of job or abnormal termination instead of issuing an EOJ and returning control to the system?

```
INADD=SYSINA
      ddname                        (MVS ONLY)
```

The ddname for the VISION:Report INA file.

```
INBDD=SYSINB
      ddname                        (MVS ONLY)
```

The ddname for the VISION:Report INB file.

```
INCDD=SYSINC
      ddname                        (MVS ONLY)
```

The ddname for the VISION:Report INC file.

```
INDD=SYSIN
     ddname                          (MVS ONLY)
```

The ddname for the VISION:Report command/table file.
You cannot change this option at run-time; this is a permanent installation option only.

```
INDDD=SYSIND
      ddname                         (MVS ONLY)
```

The ddname for the VISION:Report IND file.

```
INEDD=SYSINE
      ddname                         (MVS ONLY)
```

The ddname for the VISION:Report INE file.

```
INFDD=SYSUT1
      ddname                         (MVS ONLY)
```

The ddname for the VISION:Report INF file.

```
INGDD=SYSING
      ddname                         (MVS ONLY)
```

The ddname for the VISION:Report ING file.

```
INHDD=SYSINH
      ddname                         (MVS ONLY)
```

The ddname for the VISION:Report INH file.

```
INIDD=SYSINI
      ddname                         (MVS ONLY)
```

The ddname for the VISION:Report INI file.

```
INJDD=SYSINJ
      ddname                         (MVS ONLY)
```

The ddname for the VISION:Report INJ file.

```
INKDD=SYSINK
      ddname                         (MVS ONLY)
```

The ddname for the VISION:Report INK file.

```
INLDD=SYSINL
      ddname                         (MVS ONLY)
```

The ddname for the VISION:Report INL file.

```
INMDD=SYSINM
      ddname                         (MVS ONLY)
```

The ddname for the VISION:Report INM file.

```
INNDD=SYSINN
      ddname                         (MVS ONLY)
```

The ddname for the VISION:Report INN file.

```
INODD=SYSINO
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report INO file.

```
INPDD=SYSINP
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report INP file.

```
INQDD=SYSINQ
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report INQ file.

```
INRDD=SYSINR
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report INR file.

```
INSDD=SYSINS
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report INS file.

```
INTDD=SYSINT
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report INT file.

```
INUDD=SYSINU
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report INU file.

```
INVDD=SYSINV
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report INV file.

```
INWDD=SYSINW
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report INW file.

```
INXDD=SYSINX
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report INX file.

```
INYDD=SYSINY
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report INY file.

```
INZDD=SYSINZ
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report INZ file.

```
LIST=YES
     NO
```

List VISION:Report commands, diagnostics, and statistics on SYSLST (VSE) or SYSPRINT (MVS)? It must be stated on the first OPTION statement.

```
LISTABL=NO
        YES
```

Should VISION:Report print the table entries as they are loaded into the table?

```
LISTOPT=NO
        YES
```

List all VISION:Report options in effect? PRODCD option does not print out.

```
MBUFFER=YES
        NO                              (VSE ONLY)
```

Should VISION:Report multi-buffer all disk and tape files that space allows? (Replaces the previous OPTION NOBUFFER.)

```
MOVCVTX=NO
        YES
```

This option applies to VISION:Report statements of the format: MOVE X'....' TO field-name. NO does not convert data to target field data type. If YES, this allows automatic data type conversion based upon the target field data type, similar to the MOVEs of literal P'....' and literal C'....'.

```
MSGROLL=YES
        NO                              (MVS ONLY)
```

Should messages displayed on the system console automatically scroll off the screen without any action required by the operator? This option remains in effect for the duration of the job step.

```
OFADD=SYSUT2
      ddname                            (MVS ONLY)
```

The ddname for the VISION:Report OFA file.

```
OFBDD=SYSUT3
      ddname                            (MVS ONLY)
```

The ddname for the VISION:Report OFB file.

```
OFCDD=SYSUT4
      ddname                            (MVS ONLY)
```

The ddname for the VISION:Report OFC file.

```
OFDDD=SYSUT5
      ddname                            (MVS ONLY)
```

The ddname for the VISION:Report OFD file.

```
OFEDD=SYSOFE
      ddname                            (MVS ONLY)
```

The ddname for the VISION:Report OFE file.

```
OFFDD=SYSOFF
      ddname                            (MVS ONLY)
```

The ddname for the VISION:Report OFF file.

```
OFGDD=SYSOFG
      ddname                              (MVS ONLY)
```

The ddname for the VISION:Report OFG file.

```
OFHDD=SYSOFH
      ddname                              (MVS ONLY)
```

The ddname for the VISION:Report OFH file.

```
OFIDD=SYSOFI
      ddname                              (MVS ONLY)
```

The ddname for the VISION:Report OFI file.

```
OFJDD=SYSOFJ
      ddname                              (MVS ONLY)
```

The ddname for the VISION:Report OFJ file.

```
OFKDD=SYSOFK
      ddname                              (MVS ONLY)
```

The ddname for the VISION:Report OFK file.

```
OFLDD=SYSOFL
      ddname                              (MVS ONLY)
```

The ddname for the VISION:Report OFL file.

```
OFMDD=SYSOFM
      ddname                              (MVS ONLY)
```

The ddname for the VISION:Report OFM file.

```
OFNDD=SYSOFN
      ddname                              (MVS ONLY)
```

The ddname for the VISION:Report OFN file.

```
OFODD=SYSOFO
      ddname                              (MVS ONLY)
```

The ddname for the VISION:Report OFO file.

```
OFPDD=SYSOFP
      ddname                              (MVS ONLY)
```

The ddname for the VISION:Report OFP file.

```
OFQDD=SYSOFQ
      ddname                              (MVS ONLY)
```

The ddname for the VISION:Report OFQ file.

```
OFRDD=SYSOFR
      ddname                              (MVS ONLY)
```

The ddname for the VISION:Report OFR file.

```
OFSDD=SYSOFS
      ddname                              (MVS ONLY)
```

The ddname for the VISION:Report OFS file.

```
OFTDD=SYSOFT
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report OFT file.

```
OFUDD=SYSOFU
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report OFU file.

```
OFVDD=SYSOFV
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report OFV file.

```
OFWDD=SYSOFW
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report OFW file.

```
OFXDD=SYSOFX
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report OFX file.

```
OFYDD=SYSOFY
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report OFY file.

```
OFZDD=SYSOFZ
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report OFZ file.

```
OVLY=NO
     YES
```

Should VISION:Report run in a form of overlay mode at compile time? This option uses less memory at compile time, but causes VISION:Report to run slower during the compile phase. The execute phase time is not affected.

```
PARMEXE=NO                            (VSE ONLY)
        YES
```

Should VISION:Report use the PARM=parameter of the / / EXEC statement? If PARMEXE=YES, the contents of VAL76-179 is the same as that for MVS. Regardless of PARMEXE option, the VSE COMREG user area will be in VAL 261-271. If PARMEXE=NO, the VSE COMREG user area will also be in VAL 78-88. (See the section Val Area in Chapter 2 for VAL76-179.)

```
PARMFLD=YES
        NO                            (MVS ONLY)
```

Should VISION:Report expect a parameter field to be passed to it when dynamically invoked? If PARMFLD=YES is in effect when VISION:Report is dynamically invoked, the invoking program must pass a single parameter field, using standard linkage conventions. The field must consist of a halfword binary length followed by the parameter data itself.

For example, if you want to pass five bytes of data, you would point to a seven-byte field whose first two bytes are X'0005'. The parameter field is available to VISION:Report statements in VAL76-177.

```
PFLEOPT=NO
       YES
```

Print file counts on printer when OPTION LIST=NO is in effect?

```
PRNTLCT=54
        nnn
```

Number of lines per page on the VISION:Report report file. May be in the range of 0 through 999. Zero implies no automatic page changes. If a LINECOUNT command is present, it overrides this option.

```
PRODCOD=xxxxxxxxxxxxxxxxxxx
```

The product code assigned to your installation is issued by Computer Associates and must be used. Its length is variable and depends upon the number of CPUs and optional features you have. Normally this value is implemented when the product is installed and you should ignore this. If it is not implemented, your systems programmer (or person who installed the product) must supply you with this. If that is necessary, this option will be required for every VISION:Report execution. It is not displayed if OPTION LISTOPT=YES is specified.

```
PRTDD=SYSPRINT
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report print file. If the ddname is other than the default of SYSPRINT, it must be on the first OPTION statement.

```
PRTSIZE=133
        nnnn
```

The length or record size of the output on the VISION:Report PRT file. The first byte of the record is used by VISION:Report for standard ASA carriage control characters. MVS VISION:Report users can specify the block size on the SYSPRINT DD statement; ensure that it is a multiple of the LRECL (PRTSIZE) value.

PRTSIZE values can range from 121 to 9999. If PRTSIZE option is used, it must be on the first OPTION statement. The restriction is limited by hardware and operating system constraints.

```
PUNDD=SYSPUNCH
      ddname                          (MVS ONLY)
```

The ddname for the VISION:Report punch file.

```
PUNSIZE=81
        80                            (MVS ONLY)
```

The LRECL value (size) of the records that are output on the PUN file using the VISION:Report PUNCH statement. If PUNSIZE=81, a blank statement is always punched as the last statement to clear the punch. If PUNSIZE=80, a blank statement is not created.

```
QJMDUMP=YES
       NO
```

Should VISION:Report furnish a hexadecimal print of all active areas when canceling due to a user ABEND statement? This produces the same information as if a program check occurred.

```
RESVMEM=0
       nn                              (VSE ONLY)
```

VISION:Report reserves nnK memory. The address of this reserved area is contained in VAL257-260-B.

```
RPTDD=SYSPRINT
      ddname                           (MVS ONLY)
```

Alternate ddname for the VISION:Report report file. Default of SYSPRINT indicates that no alternate report printer is being used.

```
RPTSPCE=0
       nnn
```

Used in conjunction with the REPORT statement, RPTSPCE allows you to specify a default spacing of n to nnn between data columns. If this parameter is equal to zero, spacing is automatically determined by the REPORT logic, based upon the number and length of fields specified on the REPORT statement; if non-zero, this parameter sets the default spacing in between fields if the SPACEnn parameter is not specified.

```
RPTSYS=000
       nnn                              (VSE ONLY)
```

Alternate SYSNR for VISION:Report output. Must be between 001 and 240. The default indicates no alternate report printer requested.

```
SAVAREA=256
       nnnnnnnnnn
```

Size of the VISION:Report SAV area. May be in the range of 0 up to 2 gigabytes. This area can be used as working storage (such as in addition to WST).

```
SEQCHK=YES
       NO
```

Should the VISION:Report command statements be sequence checked? (YES requires a sequence number on each statement.)

```
SORTABL=YES
       NO
```

Should VISION:Report do an internal sort on the table entries loaded to ensure ascending sequence by argument? If NO is used, then a serial search is always done whenever an IF....ONTABLE statement is encountered. Also, if the IF...ONTABLE fails, TBH always points to the first high-value (HEX FF) entry in the table which follows the correct entries.

```
SORTPRT=NO
       YES                          (VSE ONLY)
```

Should informational messages resulting from the SORT utility be issued? SORTPRT=YES replaces SORTPRT=ALL, and SORTPRT=NO replaces SORTPRT=CRITICAL. The ALL and CRITICAL specifications are still accepted for compatibility.

```
SORTRTE=BOTH
       LST
       LOG                          (VSE ONLY)
```

Allows SORT messages to be routed to SYSLST or SYSLOG exclusively or BOTH.

```
SORTSIZ=40
       nnn                          (VSE ONLY)
```

Amount of memory for the utility SORT to use expressed in K (1024 bytes). A numeric value in the range of 0 to 9999 is valid.

```
SORTSYS=001
       nnn                          (VSE ONLY)
```

The first logical SYSNR for SORT work areas. If SORTWRK=2 and SORTSYS=005, QUIKSORT generates a control field of SORTWK=(005,006) on the SORT utility option statement.

```
SORTWRK=1
       n                            (VSE ONLY)
```

Number of SORT work areas to be used. Maximum is nine.

```
SPIE=YES
     NO                       (MVS ONLY)
```

Should program checks be trapped and a formatted print of active VISION:Report data areas be produced?

```
SQLA1=0
      9999
```

You must be licensed to use VISION:Report Interface to DB2.

This is an area of storage reserved for temporarily saving the original source statements for EXEC SQL...END-SQL statements in your VISION:Report program. If this amount of storage is exceeded during compilation, a warning message appears and VISION:Report attempts to obtain more storage and continue. If this condition consistently occurs, this option should be increased.

```
SQLA2=0
      9999
```

This is an area of storage reserved for temporarily storing the compressed EXEC SQL...END-SQL statements. Each compressed SQL takes up the length of any extended substitution variables. As an example, if the statement "... :=WST1-4095 .. " appears in an SQL statement, 4095 bytes are reserved. This area is reserved only during the compilation stage and is freed up during execution. If this amount of storage is exceeded during compilation, a warning message appears and VISION:Report attempts to obtain more storage and continue. If this condition consistently occurs, this option should be increased.

```
SQLA3=0
      9999
```

This is an area of storage reserved for temporarily storing the pointers for host variables (":") in EXEC SQL...END-SQL statements. Each host variable takes up 12 bytes. This area is reserved only during the compilation stage and is freed up during execution. If this amount of storage is exceeded during compilation, a warning message appears and VISION:Report attempts to obtain more storage and continue. If this condition consistently occurs, this option should be increased.

```
SQLA4=0
      9999
```

You must be licensed to use VISION:Report Interface to DB2.

This is an area of storage reserved for temporarily storing the pointers for extended substitution (":=") variables in EXEC SQL...END-SQL statements. Each extended substitution variable takes up 12 bytes. This area is reserved only during the compilation stage and is freed up during execution. If this amount of storage is exceeded during compilation, a warning message appears and VISION:Report attempts to obtain more storage and continue. If this condition consistently occurs, this option should be increased.

```
SQLA5=0
      9999
```

This is an area of storage reserved for holding the generated code for EXEC SQL...END-SQL statements, both during compilation and execution phases of VISION:Report.

```
Σ ((size of compressed SQL statement + size of extended host
   variables + (# host variables + # extended host variables)
   * 12) + # SQL statements * 32)
```

If this amount of storage is exceeded during compilation, a warning message appears and VISION:Report attempts to obtain more storage and continue. If this condition consistently occurs, this option should be increased.

```
SQLPLNM=********
```

You must be licensed to use VISION:Report Interface to DB2.

DB2 or SQL/DS plan name, with a maximum of eight characters.

```
SQLSYSN=****
```

You must be licensed to use VISION:Report Interface to DB2.

DB2 or SQL/DS system name, with a maximum of four characters.

```
SQLVER=nn.nn
```

You must be licensed to use VISION:Report Interface to DB2.

DB2 or SQL/DS version and release level, in the format of vv.rr. This parameter controls the generation of the dynamic SQL statements that are created from the static SQL statements embedded in your VISION:Report program. Ten character positions are available in this field. Use significant digits; no padding is necessary (for example, =2.3 or =4.1).

```
SRTADJ=NO
       YES
```

This option allows you to define the true offset relative to the start of a sort area. See SORT AREA, the SRTADJ option for more details.

```
SRTERCD=150000                                (MVS ONLY)
        nnnnnnnn
```

The estimated number of logical records that are sorted. The estimate provides better sorting efficiency especially with a large number of records involved. 1 to 8 numeric digits may be specified.

```
SRTMSG=??
       xx                             (MVS ONLY)
```

The SORT messages option expressed in two characters. Any two characters may be used that are valid with the particular SORT package that is used. The installation default prevails when omitted.

The following codes apply to most SORT systems available:

```
NO  No messages are generated.
CC  Critical messages only, routed to console.
CP  Critical messages only, routed to printer.
AC  All messages, routed to console.
AP  All messages, routed to printer.
```

```
SRTPGM=SORT
       sortname                               (MVS ONLY)
```

The program name of the utility SORT, which is loaded in to do the actual sorting.

```
SRTSIZE=0
        nnnn                          (MVS ONLY)
```

Amount of memory for the utility SORT to use expressed in K (1024 bytes). A numeric value in the range of 0 to 9999 is valid.

```
SRTWKN=NO
       YES                            (MVS ONLY)
```

Dynamic SORT work file allocation.

```
STMTEND=80
        nn
```

Allows for sequence numbers, as example, on the right side of VISION:Report statements. The nn integer is the statement column or relative position of the right hand side of all statements in the same run that delimits the end of VISION:Report statement content.

In addition, the next position immediately to the right (nn plus 1) designates whether or not the current statement is to be continued to the next statement. A statement with a non-blank character at nn plus 1 is assumed to be continued in the next statement, except when preceded by a /* comment delimiter. For simplicity, comments should end before the STMTEND column.

Continuation statements begin in column 1 and proceed long as far as required to complete a full 80 positions of the previous statement. The nn integer must be within the range of 40 through 80.

This option does not apply to any tables to be loaded. Also, any sequence numbers on the right side of VISION:Report statements do not replace or change the need for them on the left side.

As an example, code STMTEND=71 to allow sequence numbers in positions 73-80.

```
STMTIN=SYSIPT
       SYSRDR                                (VSE ONLY)
```

The name of the system logical unit that contains your VISION:Report statements and table entries, if present. SYSIPT and SYSRDR are the only valid entries.

This option cannot be overridden at run-time. It can only change on a permanent basis by recompiling the QJOPTION macro.

```
STMTLCT=50
        nnn
```

Number of lines per page on the VISION:Report print file when listing the VISION:Report command statements and diagnostics. May be in the range of 0 through 999. Zero implies no automatic page changes. There is an overhead of four lines for headings, as example, so the number specified plus four equals the true total lines per page. This option is ignored when LIST=NO is in effect. This option must be stated on the first OPTION statement.

```
STXITPC=YES
        NO                              (VSE ONLY)
```

Should program checks be trapped and a formatted print of the trace table and active VISION:Report areas be produced?

```
SUBSPIE=YES
        NO                              (MVS ONLY)
```

Should program checks that occur in a called subroutine be trapped? Due to the fact that when SUBSPIE=NO and SPIE=YES are in effect, a SPIE macro is issued before and after each VISION:Report CALL command, it is recommended that this combination of options be used only in a testing environment due to probable performance degradation. This option is ignored if SPIE=NO is in effect.

```
TRACECT=10
        nn
```

Number of VISION:Report trace entries printed on one line when TRACE ALL is in effect. Valid entries may range from 1 to 10.

```
TRLNAME=QUIKTRNT                        (MVS)
        QUKBTRN                         (VSE)
        username
```

Name of a user program that contains two translate tables for use in VISION:Report PRINTHEX and PRINTCHAR statements.

```
    1st table character line
    2nd table zone and numerics
```

```
UABNDMP=NO
        YES
```

Should a memory dump be produced when canceling due to a VISION:Report ABEND command?

```
UEXIT1=NO
       Subroutine name
```

Should a subroutine be used to retrieve VISION:Report code from a source statement library?

```
U331DMP=YES
        NO
```

Should a memory dump be produced when canceling due to I/O errors encountered?

```
U333ABE=NO
        YES
```

If YES, abend without a dump on compilation error, with code U3333. If NO, VISION:Report will issue a return code of 12, without an ABEND.

If this parameter equals NO and the DD statement has DISP=(,CATLG,DELETE), the file will not be deleted.

```
U334DMP=NO
        YES                             (MVS ONLY)
```

Should a memory dump be produced when an ABEND is due to no SYSPRINT DD statement?

```
U335DMP=YES
        NO
```

Should a memory dump be produced when VISION:Report fails to resolve run-time parameters involved in internal CALL processor?

```
U336DMP=YES
        NO
```

Should a memory dump be produced when canceling due to a program check detected by the VSE STXIT PR or MVS SPIE routines? This replaces the use of the VSE OPTION DUMP/NODUMP to determine whether a memory dump should be issued. MVS users require a SYSUDUMP or SYSABEND statement to receive a memory dump printout.

```
U338DMP=NO
       YES                              (MVS ONLY)
```

Should a memory dump be produced when an ABEND is due to a variable length record longer than the data set LRECL?

```
U339DMP=NO
       YES
```

Should a memory dump be produced when canceling due to a SORT operation failure?

```
VLABEND=YES
       NO                               (MVS ONLY)
```

Should VISION:Report ABEND U3338 if a variable length record is read or is to be written, which is longer than the data set LRECL?

```
WSTSIZE=1000
       nnnnnnnnnn
```

Size of the VISION:Report WST area. May be in the range of 0 up to 2 gigabytes.

```
XAMODE=No
       Yes
```

Size of the VISION:Report WST area. May be in the range of 0 up to 2 gigabytes.

```
ZEROPRT=NO
       YES
```

This option is only valid for release 15.0, 15.1, and 16.0 of VISION:Report. It allows a user to obtain user storage above the line starting with 16.1, this option is obsolete.

```
VSAMER=NO
       YES
```

The VSAM Error Word, controls the location in the VSAM feedback area where the VSAM error word, a 3-byte character string indicating the result of the last VSAM operation, is displayed. If set to NO then it will be displayed at VAL Area offset 253-255. If set to YES then it will be displayed at VAL Area offset 257-259. This option accommodates the 8-byte RBA returned by MVS Extended Format VSAM files. (See the description of VAL Area in Chapter 2.)

OR (Logical OR)

```
OR flddef1 WITH {flddef2 | C'xxx' | P'nnn' | X'...' | ZERO | BLANK | SPACE |
HIVALUE | LOVALUE } { flddef3 | X'...' }
```

Note: This OR verb should not be confused with the OR option on the IF verb.

Term	Description
OR	Performs the logical OR of a data field. The contents of a bit position in the source are set to one if the corresponding bit positions in either or both operands contains ones; otherwise the bit is set to zero. The resulting condition code is returned in VAL223-B. If any bits in the source operand are one following completion of the operation, then the condition code is x'01'; otherwise, it is x'00'. In the case of a ZERO figcon, if flddef1 is character, then a character zero x'F0' is used, otherwise, a binary zero x'00' is used.
flddef1	Source data. This field may be changed as a result of the operation.
WITH	Required noise word.
flddef2 C'xxx' P'nnn' X'...' ZERO BLANK SPACE HIVALUE LOVALUE	The second operand. This field is not changed.
flddef3 X'...'	Optional. A 2- or a 4-byte binary field indicating the number of bytes to be ORed. If this operand is omitted, then the length of the source field is used.

Example

```
EQU FILLER       WST0
EQU FLDA         (6)   C'ABCDEF'
EQU FLDB         (6)   X'303030303030'
    OR FLDA WITH FLDB
    PRINTHEX FLDA
    PRINTHEX VAL223

                  WST1-6 123456
                         FFFFFF
                         123456
                        01..05.

               VAL223-223
                          0
                          1
                         01
```

PAGETOTALS

```
PAGETOTALS [CUMULATIVE]
```

Note: Problems occur using PAGETOTALS if a PRINT statement is coded within a CHECKBREAKS ON BREAKS PERFORM routine.

Term	Description
PAGETOTALS	If this statement is found any place in the statement stream, the coding is generated to force a minor total print after the last detail line that is printed at the bottom of each page. This feature functions the same as a BREAK 1 level occurring once per page. PAGETOTALS is not dependent on CHECKBREAKS or BREAK statements, although these may be included for normal totaling. If you code BREAK statements, you must consider the page totals as the minor break and may also need to adjust the SPACE BEFORE and SPACE AFTER specifications. If no BREAK statement is included, one blank line appears between the last detail line and the page totals. Also, if no BREAK statements are coded, no final totals appear.
CUMULATIVE	Causes minor totals to be printed but the minor counters are not added to the next total level counters and zeroed. The net effect is that page 2 totals are the sum of page 1 totals plus the detail lines on page 2, as example.

ACCUM statements must be executed before a PRINT statement is encountered in your program. If not, the totals printed at the bottom of the page are not correct.

PAGEWIDTH

```
PAGEWIDTH nnn
```

Term	Description
PAGEWIDTH	Controls the maximum width of a report. The normal default is the value of PRTSIZE-1 (see OPTION). The PAGEWIDTH statement overrides the PRTSIZE default. This statement is a declarative.
nnn	1- to 3-digit value representing the maximum width of the report to be printed.

Example

```
PAGEWIDTH 72
PAGEWIDTH 60
```

PERFORM

```
PERFORM seq-no1 THRU seq-no2
```

Term	Description
PERFORM	Causes the statements delimited by seq-no1 and seq-no2 to be executed.
seq-no1	Sequence number of the first statement of the series of statements to be executed.
THRU	Required noise word.
seq-no2	Sequence number of an EXIT statement that represents the end of the subroutine. This operand is required and the statement must be an EXIT command on the same line as the sequence number.

The PERFORM statement:

- Transfers to the specified sequence number of the first statement of the series.

- Customizes the EXIT statement specified as the end of the subroutine to transfer to the statement following the PERFORM statement.

Coding referred to by PERFORM statements may also be executed by falling into the coding from the previous statement. The EXIT statement at the end of the routine would then deactivate itself as part of transferring back to the main part of the program and has no effect when the subroutine is fallen into from a previous statement.

PRINT

```
PRINT [DOUBLESPACE|TRIPLESPACE]
```

Term	Description
PRINT	Prints the current contents of the print line (PRT).
DOUBLESPACE	Causes a blank line before printing. This produces a double-spaced listing.
TRIPLESPACE	Causes two blank lines before printing. This produces a triple-spaced listing.

VISION:Report is set to print the number of lines specified in PRNTLCT (which can be overridden with a LINECOUNT statement) and automatically ejects to the next page after the specified number of lines and prints user defined headers, if any.

After printing, PRT is blanked.

The VISION:Report OPTION PRTSIZE is available to specify the size of the print line (PRT). The default is 133 bytes. See OPTION for more information.

PRINT REPORT

```
PRINT REPORT [DOUBLESPACE|TRIPLESPACE] [SUMMARY|[OMIT] flddef ...]
```

Term	Description
PRINT REPORT	This statement is used in conjunction with the REPORT statement. If you have not coded a REPORT statement in your program, do not use PRINT REPORT in that run. The REPORT statement is a declarative that lists all the fields that appear in your report. See REPORT for details of editing specifications or override headers. The fields stated in the REPORT statement are in the report in the same order you list them. For example, the first field is on the far left of your report; the second field is the next column. Their exact print positions are determined internally by VISION:Report. (See also RPTSPCE under the OPTION statement, and the REPORT declarative.)
DOUBLESPACE	Causes one blank line before printing your report line.
TRIPLESPACE	Causes two blank lines before printing your report line.
SUMMARY	Produces a summary report.
OMIT flddef	Suppresses printing of the specified fields.

The coding of the keyword OMIT followed by one or more field definitions suppresses the printing of those fields. The following statement prints all fields that were listed in the REPORT statement, except FLD-C and FLD-D.

```
PRINT REPORT OMIT FLD-C FLD-D
```

Using the SUMMARY operand produces a report showing only total lines for each break group. For instance, if you code a BREAK statement for DEPT, your report shows only total lines for each department, never showing the detail from each record.

The PRINT REPORT SUMMARY statement should be coded in the same place you would normally code PRINT REPORT to get a detail printed report.

PRINT REPORT SUMMARY causes the same internal MOVE TO PRT as the standard PRINT REPORT, but no printing is done. The actual print occurs only at break processing. The net result is that the print line is built each time the PRINT REPORT SUMMARY statement is handled, but at break-time the accumulated fields are moved and the entire line is printed.

Example

```
    TITLE 'SUMMARY EXAMPLE REPORT'
    REPORT FLD-A FLD-B FLD-C FLD-D FLD-E
    BREAK 1 FLD-B SB 0 SA 1
    BREAK 2 FLD-A SB 0 SA E
100 GET INF
    CHECKBREAKS
    ACCUM FLD-C
    ACCUM FLD-D
    ACCUM FLD-E
    PRINT REPORT SUMMARY
    GOTO 100
    END
```

The PRINT REPORT statement internally causes a MOVE TO PRT of each field you listed in the REPORT statement. These internal MOVES are followed by a print. By using only two statements, REPORT and PRINT REPORT, you get a completely aligned report with column headings, date, and page numbers.

The following example reads an input record and prints a line on the report for each record showing only the five fields stated in the REPORT statement.

Example

```
REPORT FLD-A FLD-B FLD-C FLD-D FLD-E     /* Define report format

010 GET INF
    PRINT REPORT                         /* Move each field to prt and print
    GO TO 010
    END
```

The PRINT REPORT statement may appear in one run any number of times. Depending on your requirements, you may need to override the fields to be printed by either:

- Coding only the fields you want printed this occurrence.

  ```
  PRINT REPORT FLD-A FLD-C            /* Print FLD-A FLD-C only
  ```

- Using the OMIT option to omit certain field(s) this occurrence.

  ```
  PRINT REPORT OMIT FLD-B             /* Print all but FLD-B
  ```

Single spacing is the default used with PRINT REPORT, but you can override that by specifying DOUBLESPACE or TRIPLESPACE. For different spacing requirements, review the SET PCC statement in this guide. When SET PCC is in effect, it becomes your responsibility to set printer spacing.

PRINTCHAR

```
PRINTCHAR flddef1 [flddef2]
```

Term	Description
PRINTCHAR	Prints the contents of the specified field in character format. This statement works like the PRINTHEX statement, except the ZONE and NUM lines are not printed.
flddef1	The VISION:Report area / definition at which printing of data begins.
flddef2	A field definition that contains a value that indicates the length of the data to be printed. This operand must be a 2- or 4-byte binary field.

The specified field is printed in 100 character increments. Any VISION:Report area may be character printed except PRT. The PRT data area is used by the PRINTCHAR statement.

The data contained in the area specified is translated using a standard EBCDIC conversion table (as example, A-Z uppercase, 0-9, and special characters) that comes with the system. If you have a private translate table, such as for upper / lowercase printing, you can invoke it by coding an OPTION statement with keyword TRLNAME=xxxxxxxx, where xxxxxxxx is the load module name (MVS) or phase name (VSE) of the private translate table.

Examples

Fixed Form:

```
010 PRINTCHAR INF1-80                 /* Print input record
020 PRINTCHAR WST43-100               /* Print working storage
```

Variable Form:

```
010 PRINTCHAR INF1 INF1-2-B           /* Length is in INF pos 1-2
100 PRINTCHAR DET1 DET1-4-B           /* Length is in DET pos 1-4
410 PRINTCHAR OFA1 WST1-2-B           /* Length is in WST pos 1-2
```

Warning: If the specified length is in error and causes VISION:Report to go beyond its area, region, partition, as examples, program checks could occur.

PRINTHEX

```
PRINTHEX flddef1 [flddef2]
```

Term	Description
PRINTHEX	Prints the contents of the specified field in hexadecimal, character, and numeric format.
flddef1	The VISION:Report area / definition at which hex printing of data begins.
flddef2	A field definition that contains a value that indicates the length of data to be hex printed. This operand must be either a 2- or 4-byte binary field.

The described field is printed in 100 character increments. Any VISION:Report area may be hex printed, except PRT. The PRT data area is used by the PRINTHEX statement.

The data contained in the area specified is translated using a standard EBCDIC conversion table (for example, A-Z uppercase, 0-9, and special characters) that comes with the system. If you have a private translate table, such as for upper / lowercase printing, you can invoke it by coding an OPTION statement with keyword TRLNAME=xxxxxxxx, where xxxxxxxx is the load module name (MVS) or phase name (VSE) of the private translate table.

Example

Fixed Form:

```
010 PRINTHEX INF1-80                  /* Print input record
020 PRINTHEX WST43-100                /* Print working storage
```

Variable Form:

```
010 PRINTHEX INF1 INF1-2-B            /* Length is in INF pos 1-2
100 PRINTHEX DET1 DET1-4-B            /* Length is in DET pos 1-4
410 PRINTHEX OFA1 WST1-2-B            /* Length is in WST pos 1-2
```

Warning: If the specified length is in error and causes VISION:Report to go beyond its area, region, partition, as examples, program checks could occur.

PUNCH

```
PUNCH flddef
```

Term	Description
PUNCH	Writes the contents of the data area PUN to the card punch using symbolic unit SYSPCH in VSE and ddname SYSPUNCH in MVS. The PUN data area is then blanked by VISION:Report.
flddef	The area / definition at which punching of data starts.

Example

If the field INF1-450 is coded, then 6 records are written.

```
Record 1    INF1-80
Record 2    INF81-160
Record 3    INF161-240
Record 4    INF241-320
Record 5    INF321-400
Record 6    INF401-450 followed by 30 blanks
```

In the previous example, after the PUN data is used, its contents are destroyed using the PUNCH INF1-450 form of the PUNCH statement.

MVS users can change the record size for SYSPUNCH by using the PUNSIZE option (see OPTION). This option allows the record to be 80 bytes long without the punch control character, plus a blank statement is not punched when EOJ is reached.

READ

```
READ {INA-INZ|DET} USING flddef {EQUAL|GENERIC} [ONERROR seq-no]
```

Term	Description
READ	Randomly retrieves a record from a KSDS, ESDS, or RRDS VSAM file based upon the key field specified.
INA-INZ DET	File name can be any VISION:Report input file name (INA-INZ, DET).
USING	Required noise word.
flddef	Field definition or an equated data name that contains the key of the record desired. If the file is KSDS, the length of this area cannot be less than the key length defined for it, with padded fill characters to the right if necessary. If the file is ESDS or RRDS, the length of this area must be 4 bytes binary. The key for an ESDS VSAM file is its RBA (relative byte address), whereas an RRDS key is the relative record number of the desired record.
EQUAL GENERIC	The type of match desired between the key and the file. The EQUAL operand indicates that a record with an equal key is to be returned, whereas the GENERIC operand indicates that a record with an equal key or, if an equal key cannot be found, a record with the next greater key is to be returned. If neither is specified, EQUAL is the default. The GENERIC operand cannot be specified for ESDS or RRDS VSAM files.
ONERROR seq-no	Causes VISION:Report to automatically transfer to the statement with the specified sequence number when a VSAM error occurs. This allows you to examine the error codes (RC/EC) and take appropriate action. If this operand is not coded and a VSAM error occurs, execution continues with the next statement, and it is your responsibility to check for these errors.

The length of the record is returned in the 2-byte or 4-byte length field preceding the I/O area and in VAL243-246-B. To access the length field, see the techniques shown in the section VSAM Recommendations in Chapter 2.

The following error is returned when the ONERROR operand is not coded:

	Error Word	RC/EC
Error	VAL253-255	VAL247-248
RECORD NOT FOUND	'RNF'	X'0810'

RELEASE

```
RELEASE flddef [TO SORT]
```

Term	Description
RELEASE	Specifies the location of a record for sorting. This statement is used in conjunction with the SORT AREA statement. You cannot use RELEASE with a SORT FILE specification.
flddef	The starting location (first position) of the record. This operand must be an equated data name or VISION:Report field definition.
TO SORT	Required noise words. If this operand is included, it must be coded as shown.

The reuse or execution of the RELEASE statement automatically invokes a new SORT operation, providing an intervening RETURN statement has been encountered.

Example

You have a very large Accounts Receivable file with a record size of 352 bytes. However, you only want to sort the zip code, customer name, and street fields of records with an account code of MO. Use the SORT option to efficiently manage both machine time and the SORT work area size. Using SORT, select only those records you want, build an area for the selected information, and pass these shortened records to SORT.

```
EQU MY-AREA        WST1-80            /* Entire sort area
    EQU ZIP        WST1-5             /* Zip Code
    EQU CUST-NAME  WST6-30            /* Name
    EQU STREET     WST31-55           /* Street
    REPORT ZIP  CUST-NAME   STREET
    SORT AREA (F) RL80 ON ZIP  CUST-NAME   STREET
    MOVE SPACES             TO MY-AREA
1   GET INF ATEND 2                   /* Original record size 352 bytes
    IF INF182-183  IS NOT EQ C'MO'    /* Select only "MO" code
        GOTO 1.
     MOVE INF155-159         TO ZIP   /* Build sort work area of 80 bytes
     MOVE INF85-109          TO CUST-NAME
     MOVE INF110-134         TO STREET
     RELEASE WST1 TO SORT             /* Turn record to be sorted over to sort
     GO TO 1 /* Get another record from input file
2    RETURN SORTED INTO WST1 ATEND EOJ /* Start returning records
* INSTEAD OF THE "ATEND 3" CLAUSE ABOVE,
* COULD HAVE CHECKED FOR VAL200 = 'E' INSTEAD
     PRINT REPORT
     GO TO 2
9999 END
```

For an additional example, see RETURN.

REPORT

```
REPORT flddef [SPACEnn] [nC|nE|nN|n mask-code] [(override-headers)] [flddef ...]
```

Term	Description
REPORT	Specifies the fields to appear in the report generated by VISION:Report's automatic report writer. This statement is a declarative. You can specify spacing, override headers, and edit masks. These optional operands may be interspersed within the declarative in any order, so long as override headers and edit codes follow the REPORT data name field.
flddef	Up to 32,000 VISION:Report field definitions or equated names may be specified to appear in the report line. When the report line is longer than the option PRTSIZE, a diagnostic occurs. If the accumulators CTA through CTP are used with the ACCUM statement, the REPORT declarative must also specify the field definition or equated name with the proper number of bytes addressed by the rightmost positions. (See ACCUM.)
SPACEnn	Allows control of spacing between data columns, where nn is a 1- or 2-digit number stating the number of spaces between the report columns. SPACEnn should be stated between two field definitions or preceding the first field definition in the REPORT declarative. The following example specifies that the report contains FLD-A, FLD-B, FLD-D, and FLD-F. It generates ten spaces between FLD-D and FLD-F. SPACE00 signifies that no spacing is to take place. `REPORT FLD-A FLD-B FLD-D SPACE10 FLD-F` When SPACEnn is not specified, VISION:Report automatically generates spacing for you, with the default spacing allowing the report to be centered based upon the number and length of each field.
If RPTSPCE=0, VISION:Report automatically generates spacing for you with the default spacing allowing the report to be centered based upon the number of length of each field. If RPTSPCE=nn (a non-zero number) is specified, then VISION:Report automatically generates a SPACEnn between each field.	

Term	Description

mask-code	Defines the edit mask code.
	C Commas are inserted as appropriate.
	E European variation of commas / decimals appear.
	N Zero suppression is not performed.
	n Signifies how many digits are to appear to the right of the decimal point. Any number in the range of 0 through 9 may be coded.
	`INF1-8 2C may produce 999,999.00`
	Any VISION:Report or user-supplied edit masks may be applied to the reported data fields. The following example produces a report with two override headers and three VISION:Report generated headers. The phone field is edited with the phone number mask.
	`REPORT INF1-30 (NAME) INF40-49 (PHONE-NUMBER) P`
	`WORK-NUMBER CLIENT-NUMBER WST4-6`

When OPTION EDIT=YES is specified (to compile, but not execute your program), you get a proof copy of your report.	

Term	Description

override headers	Heading written following the data names they head. Any number of spaces (including none) may fall between the header and the data name. This operand can be up to 50 characters and must be enclosed in parentheses. If headers are not specified, VISION:Report generates column headings from the equated names and field definitions in the REPORT declarative.
	
	`INF1-20 (NAME)  INF21-30 (ADDRESS)`
	
	The above example causes a heading of NAME above the data column containing INF1-20, and ADDRESS above the data column of INF21-30. The heading is centered or right-aligned in case of numerics, and the fill characters used are periods. (The HDRDOTS option allows suppression of these periods.) Commas, spaces, and hyphens (in order of priority) force headings into multiple lines.
	
	`(CUSTOMER NAME-AND ADDRESS) composes as CUSTOMER NAME-AND ADDRESS`
	`(CUSTOMER-NAME-AND-ADDRESS) composes as CUSTOMER NAME AND ADDRESS`
	`(T, I,  T,  L,  E ) composes as  T I T L E`
	
	If override headers exceed 50 characters, a diagnostic is issued. A line of headers that exceeds the option PRTSIZE is trimmed (one character from each header) until it fits within that print line.

REPORT can be stated once followed by the report fields or can be repeated at the beginning of each line of report fields. Either format produces the same report layout. VISION:Report builds the print line assigning positions to each data name in the same order as in the REPORT statement. For example, your first data name appears on the left side of the page, your second data name appears next.

The following two examples produce the same report.

```
REPORT FLD-A FLD-B
     FLD-E FLD-X FLD-Y
```

or

```
REPORT FLD-A FLD-B
REPORT FLD-E FLD-X FLD-Y
```

RETURN

```
RETURN SORTED INTO flddef [ATEND {EOJ|seq-no}]
```

Term	Description
RETURN	Receives a sorted record and the location to where it is to be delivered. This statement is used in conjunction with the SORT AREA statement. The first time RETURN is executed, the records that have been turned over for storage by the RELEASE statement are sorted. The first sorted record is returned or made available for use. The second and subsequent times RETURN is executed, the next sorted record is returned or made available for use, or you are informed there are no more sorted records to process.
SORTED INTO	Required noise words.
flddef	The starting location (first position) of the record to be returned, and must be an equated data name or VISION:Report field definition.
ATEND	Controls when the file reaches end of file.
seq-no EOJ	Sequence number of statement to receive control when end of file for the file occurs. When end of file does occur, VISION:Report transfers control to the statement specified by seq-no. EOJ can also be coded in place of the sequence number, causing VISION:Report to transfer to its end of job routine.

If the ATEND seq-no operand is coded, you must include a GOTO EOJ statement in your EOJ routine to allow VISION:Report to perform operations such as, close all files or take final totals.

Then the last record has been returned, an end of file on sorted records is indicated in two ways:

- VAL area position 200 is set to EBCDIC 'E' (hex C5).
- The RETURN area is filled with high-values (hex FF).

You can test either of the areas with an IF statement or use the ATEND operand to automatically transfer to either the statement whose sequence number is nnnn or EOJ.

Examples

```
010 RETURN SORTED INTO WST1
020 IF VAL200 IS EQ C'E'
030  GOTO EOJ.
```

This example varies slightly from the example shown in the RELEASE statement section. This example selects the state, account code, and customer name for customers in the state of California and Illinois. The example breaks on state and account code and accumulates all installment payments. The printed report shows the customer's state, account code, name, and the current installment payments.

```
EQU  MY-AREA        WST1-37              /* Entire area
     EQU STATE      WST1-2               /* State code
     EQU CUST-NAME  WST6-30              /* Name
     EQU ACCT-CODE  WST31-32             /* Account code
     EQU INSTL-PAY  WST33-37-P  2C       /* Installment payment
     REPORT  ACCT-CODE  STATE  CUST-NAME  INSTL-PAY
     SORT AREA (F) RL37 ON STATE  ACCT-CODE  CUST-NAME
     BREAK 1  STATE         SB 1 SA 2 PRINT C'STATE TOTALS'
     BREAK 2  ACCT-CODE     SB 1 SA 2 PRINT C'ACCT  TOTALS'
     BREAK F  STATE  SB 2 SA 2 PRINT C'FINAL TOTALS' IN TOT POS 35
     MOVE SPACES             TO MY-AREA
100  GET INF ATEND 200
     IF INF153-154  IS     EQ  C'CA'     /* Select California
        GOTO 150.
     IF INF153-154  IS NOT EQ  C'IL'     /* Select Illinois
        GOTO 100.
150  MOVE INF153-154      TO STATE       /* Build sort work area
     MOVE INF85-109       TO CUST-NAME
     MOVE INF182-183      TO ACCT-CODE
     MOVE INF197-201-P    TO INSTL-PAY
     RELEASE WST1 TO SORT                /* Turn record to be sorted over to sort
     GO TO 100                           /* Get another record from input file
200  RETURN SORTED INTO WST1             /* Start returning records
     IF @VAL-SORT-EOF  EQ C'E'           /* EOF on sort input
        GOTO 300.
     CHECKBREAKS
     ACCUM INSTL-PAY  IN A 5 BYTE CTA ON BREAKS PRINT IN POS 47 2C
     PRINT REPORT
     GO TO 200
300  MOVE C'END OF TEST' TO PRT1
     PRINT TRIPLE SPACE
     GO TO EOJ
9999 END
```

REWRITE (VSAM Only)

```
REWRITE {INA-INZ|DET} [ONERROR seq-no]
```

Term	Description
REWRITE	Updates the last record retrieved from a VSAM file.
INA-INZ DET	File name can be any VISION:Report input file name (INA-INZ, DET).
ONERROR seq-no	Causes VISION:Report to automatically transfer to the statement with the specified sequence number when a VSAM error occurs. This allows you to examine the error codes (RC/EC) and take appropriate action. If this operand is not coded and a VSAM error occurs, execution continues with the next statement, and it is your responsibility to check for these errors.

The length of the record was placed in the record length field preceding the I/O area and in VAL243-246-B when the record was retrieved. If the length of the record is to be changed, see the techniques shown in the section VSAM Recommendations in Chapter 2.

REWRITE (ISAM Only)

```
REWRITE {INA-INZ|DET}
```

Term	Description
REWRITE	Writes the contents of a file record area to the appropriate input ISAM file. The input must be an ISAM file. Unpredictable results occur if the REWRITE is used for other file types. Records may not be added to an ISAM file.
INA-INZ DET	File name can be any VISION:Report input file name (INA-INZ, DET).

Examples

```
010 GET INF
020 MOVE ZEROES TO INF95-100-P          /* Move packed zeros to INF95-100
030 REWRITE INF                         /* Update the file in place
040 GO TO 010                           /* Go get the next input record

010 GET INF                             /* Get an input record
020 IF INF95-100-P IS NOT NUMERIC       /* If INF95-100-P is not numeric
030    MOVE ZEROES TO INF95-100-P       /* Move packed zeros to INF95-100
040    REWRITE INF.                     /* And update the file in place
050 GO TO 010                           /* Go get the next input record
```

The previous example reads the entire INF file. Each record is inspected to see if field INF95-100 (packed data) is numeric. The records that are not numeric in INF95-100-P are cleared to packed zero and updated (rewritten) to the INF file.

A rewrite of one record in a block causes the entire block to be rewritten. Also, ISAM records are processed in GET LOCATE mode, not in a work area. Thus, if a record is modified, but no REWRITE is issued, the record is rewritten if any record in the same block is rewritten.

SAMPLE

```
SAMPLE nnnnnnn {INF|INA-INZ|DET}
```

This statement is declarative and therefore cannot be the subject of a transfer in control statement (GOTO, PERFORM, as examples).

Term	Description
SAMPLE	Causes every nth record from the specified file to be passed in response to a GET statement.
nnnnnnn	The sample level desired. If you enter 10, the first GET returns the first record for the specified file, the second GET returns the eleventh record, the third GET the twenty-first record, and so on.
INA-INZ DET	File name can be any VISION:Report input file name (INA-INZ, DET). If a file name is not given, INF is assumed.

If a LIMITREADS statement is present for the specified file, only SAMPLE records passed decrement the LIMITREADS count.

When using multiple input files, you are responsible for determining end of file and EOJ. See GET for end of file processing.

Example

```
010 SAMPLE 100 IND
020 GET IND ATEND EOJ
030 MOVE IND1-85 TO PRT1
040 PRINT
050 GOTO 010
```

In this example, list the IND file to determine if a previous job built the records correctly. Rather than list the first part of the file, sample it by listing only records 1, 101, 201, 301, as examples.

If the file is extremely large, you might include a LIMITREADS statement to limit the number of records printed. See LIMITREADS for an example.

SET

```
SET {PTA-PTH|PTR} flddef
SET {PTA-PTH|PTR} {SAVE|RESTORE} [flddef]
SET {PTA-PTH|PTR|TSA|TBH} {UP|DOWN} {value|flddef}
SET {TSA|TBH} INITIAL
```

Term	Description
SET	Initialize or modify the address of a general or special purpose pointer.
PTA-PTH	Eight general purpose pointers.
PTR	Special purpose pointer altered following a match on an IF ... INCLUDES or a TRNT statement.
TSA	Special purpose pointer to the first table entry used by an IF ... ONTABLE statement when searching for a match.
TBH	Special purpose pointer to the matching table entry following a hit on an IF ... ONTABLE statement.
action	Enter the initialization or modification action that is to be performed on the address in the pointer specified. The various allowable entries have the following effects: `<none>` — Sets the pointer to a flddef. INITIAL — Sets the TSA pointer to the first table entry or the TBH pointer to the matching table entry following the most recent hit on an IF … ONTABLE statement. RESTORE — Sets the pointer to an address specified in a 4-byte binary flddef or to the value saved in a reserved internal location by a previous SAVE statement. SAVE — Stores the address of a pointer in a 4-byte binary flddef or in a reserved internal location. DOWN — Negatively adjusts the address of a pointer by a value or by the value specified in a flddef. UP — Positively adjusts the address of a pointer by a value or by the value specified in a flddef.
value	A decimal number that a pointer is going to be adjusted by.
flddef	A storage location containing the value that a pointer is going to be set to or adjusted by.

SET PTx SAVE|RESTORE without a flddef saves to or restores from an internal area that is not addressable by the user. Only one value can be held at a time. To save or restore multiple values, or if the value is to be addressable by the user, specify a flddef.

The obsolete statement SET PTx PTR VALUE, which is the equivalent of SET PTx PTR1, is maintained for upward compatability.

Set Allowable Forms:

- TSA and TBH can be set to initial.
- PTA-PTH, PTR, TSA, and TBH can be set up or down by a number or flddef.
- PTA-PTH and PTR can be set to flddef.
- PTA-PTH and PTR can be SAVED or RESTORED.

Warning: No checking is done by VISION:Report to ensure that pointer manipulations are within range of the intended data, within the partition or region, as examples. This is your responsibility.

Also, review the warning in MOVE.

SET PCC

```
SET PCC {flddef|SINGLESPACE|DOUBLESPACE|TRIPLESPACE|ZEROSPACE TOP-OF-FORM|C'x'|OFF}
```

Term	Description
SET PCC	This addresses and allows the overriding of the printer carriage control character, sometimes referred to as the ASA control character. See the description of C'x' in the following table for a listing of the printer carriage control characters. This overrides any spacing operands used with a PRINT statement (including PRINT REPORT). However, PRINTCHAR, PRINTHEX, DOHEADERS, and PAGETOTALS override any PCC settings. PCC is a reserved pointer name to identify the printer carriage control character.
flddef	This defines a standard VISION:Report field definition or EQU name whose contents are to be moved to the printer carriage control. (Length must be one character; see C'x' for allowable forms.)
ZEROSPACE	Do not space any lines before printing, print over top of the last line printed.
SINGLESPACE	Space one line before printing.
DOUBLESPACE	Space two lines before printing.
TRIPLESPACE	Space three lines before printing.
TOP-OF-FORM	Space to the top of the next page before printing. This does not automatically cause header printing or increment the VISION:Report page number.
C'x'	Must be one of the following valid ASA carriage control characters: blank — Single-spacing 0 — Double-spacing - — Triple-spacing + — Suppress spacing 1-9 — Skip to channel 1-9, respectively A,B,C — Skip to channel 10, 11, 12 respectively
OFF	Turns the user PCC pointer off, and allows the operands DOUBLESPACE and TRIPLESPACE in a following print statement to be recognized.

Example

```
110 GET INF ATEND 250              /* INF1 Contains carriage control
120 SET PCC INF1                   /* Character for record
130 MOVE INF2-80 TO PRT1           /* Move record to print
140 PRINT                          /* Print record
150 GO TO 120                      /* Get next record
250 SET PCC TOP-OF-FORMS           /* Skip to top of page
260 MOVE C'SUMMARY PAGE' TO PRT30  /* For summary page
270 PRINT
280 SET PCC C'+'                   /* Over-print on summary
290 MOVE C'_________' TO PRT30     /* Page underline to make
300 PRINT                          /* Heading stand out, turn
310 SET PCC OFF                    /* Off PCC before continuing
```

SETGENKEY (VSAM Only)

```
SETGENKEY {INA-INZ|DET} USING flddef {EQUAL|GENERIC nn} [ONERROR seq-no]
```

Term	Description
SETGENKEY	Positions a VSAM file so that the next GET statement returns a record with a key equal to the specified field definition or a key equal to or greater than the specified field definition.
INA-INZ DET	File name can be any VISION:Report input file name (INA-INZ, DET).
USING	Required noise word.
flddef	A field definition or an equated data name that contains the key of the record at which it is to be positioned. If the file is ESDS or RRDS, this area must be 4 bytes binary. The key for an ESDS VSAM file is its RBA (relative byte address), whereas an RRDS key is the relative record number of the desired record. If the file is KSDS and the GENERIC operand is not used, the length of this area must be the same as the key length defined for it.
EQUAL	Positions the file at the record with a key equal to the contents of the field definition, if one exists. The GENERIC operand positions the file at the record with a key equal to or greater than the contents of the field definition. This operand cannot be used on ESDS or RRDS VSAM files. EQUAL is the default.
GENERIC nn	Positions the file at the record with a key equal to or greater than the contents of the field definition. This operand cannot be used on ESDS or RRDS VSAM files. The nn must be in the range of 01-99, indicating the length of the key to be used in positioning the file.
ONERROR seq-no	Causes VISION:Report to automatically transfer to the statement with the specified sequence number when a VSAM error occurs. This allows you to examine the error codes (RC/EC) and take appropriate action. If this operand is not coded and a VSAM error occurs, execution continues with the next statement. It is your responsibility to check for these errors.

The following error is returned when the ONERROR operand is not coded:

	Error Word	RC/EC
Error	VAL253-255	VAL247-248
RECORD NOT FOUND	'RNF'	X'0810'

SETGENKEY (ISAM Only)

```
SETGENKEY {INA-INZ|DET} USING flddef
```

Term	Description
SETGENKEY	Causes the next GET from an ISAM file to be the first record at the generic key class pointed to by the field definition.
INA-INZ DET	Enter the file name to indicate for which specified file the pointers are to be initialized.
flddef	Define the field (area and starting position) that contains the generic key to be used in positioning the file. This field should be the same length as the key of the ISAM file. The key placed in this field should be appropriately padded with blanks or binary zeros. An actual full length key may be used.

It is important to test the key of returned records after a GET statement to ensure that SETGENKEY has found the beginning of the key class and presumably to determine when the end of the key class has been reached.

The SETGENKEY, GET, and REWRITE statements may be combined to retrieve, update, and rewrite individual records.

Keys of less value than the lowest in the file cause the first record in the file to be retrieved.

Keys of greater value than the highest in the file cause VISION:Report to:

- Place high-values (HEX FF) in the record area, and
- Place an E in the appropriate EOF byte position in the VAL area. (Another SETGENKEY for the same file resets the E to a space.)

SETGENKEY is designed to provide sequential processing starting at some point other than the beginning of the file.

As a means of random processing, QUIKISAM is recommended. SETGENKEY is not recommended. QUIKISAM is faster and provides precise hit or no hit indicators.

The following example shows how to read transactions from the detail file, find their matches in the INF (ISAM) file, update the INF record, and rewrite it in place. The key is assumed to be 5 positions, starting in position 1 of the detail record.

Example

```
010 GET DET                          /* Get a detail record
020 IF DET1-5 IS HIVALUE             /* Test for end of transactions
030      GOTO EOJ.                   /* Go to end of job on EOF
040 SETGENKEY INF USING DET1         /* Do the key set
050 GET                              /* Do get on ISAM
052 IF VAL196 IS EQ C'E'             /* Test if key beyond end of file
054      GOTO 010.                   /* Yes, go get another DET record
060 IF DET1-5 IS EQ TO INF1-5        /* If keys are equal
070      PERFORM 300 THRU 400        /* Perform maintenance routine at
075      *                           /* 300-400
080      GOTO 010.                   /* Go get next transaction
090 IF INF1-5 IS LT DET1-5           /* If ISAM is less than transaction
100      GOTO 050.                   /* Go get an ISAM record
110 * Master is greater-Do NO HIT coding
120 GOTO 010
300 * Maintenance routine
310 * Move detail transaction data to INF file record
320 REWRITE INF                      /* Update ISAM
400 EXIT
```

SKIP

```
SKIP {flddef|nnnn} {INA-INZ|DET|OUTPUT} [ATEND {seq-no|EOJ}]
```

Term	Description
SKIP	Causes nnnn number of records to be skipped (or ignored). This statement may refer to any of the input files or the report output. When the SKIP statement refers to the report output, it causes nnnn number of print lines to be skipped, thus spacing down the page.
nnnn flddef	Enter the actual number of records (or print lines) to skip, or the name of a VISION:Report field definition containing the number. Flddef must be a 2- or 4-byte binary area. nnnn is any number from 1 to 2 gigabytes.
INA-INZ DET OUTPUT	Enter either the keyword OUTPUT or any input file name. The keyword OUTPUT is associated with the printed output.
ATEND	Controls when the file reaches end of file.
seq-no EOJ	Sequence number of statement to receive control when end of file for the file occurs. When end of file does occur, VISION:Report transfers control to the statement specified by seq-no.

If the ATEND seq-no operand is coded, you must include a GOTO EOJ statement in your EOJ routine to allow VISION:Report to close all files, take final totals, as examples.

When end of file occurs, VISION:Report:

- Moves high-values (X'FF') to the file's record area.
- Moves an E to the appropriate position in the VAL area.

Examples

```
SKIP 10 OUTPUT                  /* Space down 10 print lines on report
SKIP 150 INF ATEND EOJ          /* Read and ignore 150 records of INF
```

Sorting

VISION:Report has three different sorting options:

- File sorting (sort one of the VISION:Report input files).
- Area or record sorting (select, match, and create records to be sorted).
- An immediate in-place sort of a VISION:Report area of storage.

The VISION:Report SORT feature functions much the same as the COBOL sort statement, and invokes the installation's sort utility to do the actual sorting of records. VISION:Report interprets the SORT statement or SORT FILE or SORT AREA declaratives and builds the necessary internals for the sort. To use the SORT feature, you must have an IBM or IBM compatible SORT installed on your system.

SORTWKxx files need not be specified in JCL for MVS. SORT uses the dynamic allocation of work files. SORT control statements use the following convention:

Statement convention		
File Sort	Q+filename	for example, QINF
Area Sort	QARE	
Sort Verb	QARV	

Your installation could require SORTLIB and SYSOUT DD statements, if MVS.

VSE SORTWKn will be required in the same manner (and number) as when invoking a standard sorting operation.

The following table lists the SORT options. They may be included in a VISION:Report at execution time and/or permanently made an installation option with the QJOPTION macro.

Option	Description
SRTSIZE=nnnn	Amount of memory for the utility SORT to use expressed in K (1024 bytes). The installation default memory size is used when omitted or zero. A numeric value in the range of 0 to 9999 is valid. Zero is the default.

Option	Description
SRTWKN=NO YES	VISION:Report can sort multiple files or areas within the same program concurrently if YES option is chosen. ■ **MVS:** To do this, VISION:Report dynamically allocates input and sort files. The filename consists of 'Q' followed by three letters signifying the file or area, such as INF, DET, ARE (SORT AREA) or ARV (SORT VERB). Work areas are also dynamically allocated. ■ **VSE:** Since files cannot be dynamically allocated, the proper DLBL and EXTENT statements must be coded.
SRTERCD=nnnnn	The estimated number of logical records to be sorted. The estimate provides better sorting efficiency especially with a large number of records involved. You must specify 1 to 8 numeric digits. 150000 is the default.
SRTPGM=sortname	The program name of the utility SORT which VISION:Report loads to do the actual sorting. The default program name SORT is used when omitted.
SRTADJ=NO YES	The SRTADJ option allows you to define the true offset relative to the start of an area. See SRTADJ Option. NO is the default.
SRTMSG=xx	MVS only. The SORT messages option expressed in two characters. Any two characters may be used that are valid with the particular SORT package that is used. The installation default prevails when omitted. The following message codes apply to most SORT packages available. Some program product sort routines allow codes in addition to those listed below. SRTMSG= NO No messages are generated CC Critical messages only, routed to console CP Critical messages only, routed to printer AC All messages, routed to console AP All messages, routed to printer
U339DMP=YES NO	Should dump be produced when abending due to a SORT operation failure? NO is the default.

Note: For more information on SORT parameters, see the section ADDITIONAL SORT OPTIONS.

SORT Fields

```
SORT flddef1 [flddef2] RLnnnnn ON flddef [(D)] [flddef [(D)] ...]
```

Term	Description
SORT	Performs an immediate in-place ordering of records in a VISION:Report area of storage using the following parameters.
flddef1	Specify the area of storage to be sorted. If the number of records to be sorted does not vary during the course of program execution, include the starting and ending offsets of the area. Otherwise, include only the starting offset of the area. The area size and the RLnnnnn parameter determine the number of records. The area size must be evenly divisible by the record length.
flddef2	Optional. If flddef1 is defined including only a starting offset, then include this parameter. Specify a 2- or 4-byte binary field containing the number of records to be sorted. Each time the SORT statement is executed, VISION:Report uses the value in this field definition to determine the number of records to be passed to the sort utility.
RLnnnnn	Enter RLnnnnn, where nnnnn is the length of your fixed length records. RL may be stated with a 1- to 5-digit number.
ON	Required noise word.
flddef	Enter the equated data name or field definition of the field on which you want the major sort sequence. If you define more than one sort field, each subsequent field is a more minor sort sequence than the last (for example, the most major sort sequence is the first field, then the second).
(D)	If the previous field is to be sorted into descending sequence, enter (D). Otherwise, enter the equated data name or field definition of the field you desire to be sorted next (if any). The default is ascending sequence.

Examples

Using a single field definition specifying a starting and ending offset:

```
EQU  WA-TABLE        WST5001-5030
EQU  KEY1            WST5002-5003
EQU  KEY2            WST5005-5006-B
EQU  KEY3            WST5008-5010-P
EQU  WA-TABLE                                    /* Redefines
EQU  REC1            (10) X'00C1C20012340012345C'
EQU  REC2            (10) X'00C1C20056780012345C'
EQU  REC3            (10) X'00C4C50056780012345C'

SORT WA-TABLE RL10 KEY2 (D) KEY3 KEY1
```

Using two field definitions, one specifying a starting offset only and a second with the value of the number of records to be sorted.

```
EQU  WA-PTR          WST5001
EQU  KEY1            WST5002-5003
EQU  KEY2            WST5005-5006-B
EQU  KEY3            WST5008-5010-P
EQU  WA-RECS         WST6101-6104-BØX'00000003'
EQU  WA-PTR                                      /* Redefines
EQU  REC1            (10) X'00C1C20012340012345C'
EQU  REC2            (10) X'00C1C20056780012345C'
EQU  REC3            (10) X'00C4C50056780012345C'

SORT WA-PTR WA-RECS RL10 ON KEY2 (D) KEY3 KEY1
```

An example of using SORT fields and TABLSPEC is in the distributed examples member TBLSRT2.

SORT AREA

```
SORT AREA {(F)|(V)} RLnnnnn ON flddef [(D)] [flddef [(D)] ...]
```

Term	Description
SORT AREA	Sorts only a portion of a file (for example, 10 percent). Area sorting also allows you to match data from any number of input files (for example, VSAM files, ISAM files, IMS, DL/I), pick up extra data and build a record format that suits your purpose, and then sort the records into the sequence you need for reporting, processing, writing sorted output, as examples.
(F), (V)	If record format is variable length records, enter (V). Fixed length records are assumed. However, you may code (F) to indicate fixed length records.
RLnnnnn	Enter RLnnnnn, where nnnnn is the length of your fixed length records, or in the case of variable length records nnnnn must be the maximum length of the records you want to process. (RL may be stated with a 1- to 5-digit number.)
ON	Required noise word.
flddef	Enter the equated data name or field definition of the field on which you want the major sort sequence. If you define more than one sort field, each subsequent field is a more minor sort sequence than the last (for example, the most major sort sequence is the first field, then the second).
(D)	If the previous field is to be sorted into descending sequence, enter (D). Otherwise, enter the equated data name or field definition of the field you want to be sorted next (if any). The default is ascending sequence.

The SORT AREA statement also requires the use of the RELEASE and RETURN statements.

Examples

```
SORT AREA RL100 ON WST21-29 WST161-164-P (D) WST30
SORT AREA (V) RL300 ON INF7-12 INF16-B INF14-15-B
SORT AREA RL200 ON PTA21-26-P (D)
SORT AREA RL80 ON ORDER# ORDER-DATE                /* EQU data name
```

The SORT statement has the following limitations:

- All operands must be on one statement.
- Sum of all SORT fields may not exceed 256 bytes.
- The work area for the sort control statements is 512 bytes. As many fields may be specified as will fit in this area.
- File sorts must either be fixed or variable record types.

The VISION:Report field definitions or equated data names referenced in the SORT AREA statement are assumed to be relative to the first position of each logical record (AREA), unless the SRTADJ option is used. (See SRTADJ Option.)

If you want to SORT records on positions 1 through 5 and RELEASE the record at WST251, the SORT AREA statement requires that WST1-5 be specified as the SORT control field.

To resolve this, specify PTn1-5 as the SORT control field, and RELEASE the record to SORT as PTn1.

```
SET PTn WST251
```

To SORT records that are of variable length or undefined until the point of use:

- Specify variable (V) on the SORT AREA statement. You must specify a record length sufficiently large enough to hold the maximum record you will ask the SORT to handle.
- Move to VAL-201-204-B the length of each record prior to executing the RELEASE statement.

VISION:Report uses this length to gather the record, from the specified location, into an internally generated work area. The record is converted to IBM standard variable length record (RL00DATA-DATA-DATA) prior to turning over to the SORT.

When you ask for the record to be returned, the RL00 is stripped off. The data portion is moved into the area you requested. VAL201-204-B is not required and is ignored on file sorting and fixed length record sorting.

When you ask for the record to be returned, the record length (in binary) is placed in VAL201-204-B on variable length record sorting only.

If you are going to sort existing true IBM variable style records in the RL00DATA-DATA format, the RL portion includes the 4 bytes occupied by RL00 as part of the length found stored in the RL portion.

Data starts in position 5. This has an effect on what position you specify a field to start in.

The minimum record length allowable using SORT AREA (V) is 18 bytes.

SRTADJ Option

The option SRTADJ makes it easier to code SORT declarations. This option allows you to define the true offset relative to the start of an area. If SRTADJ=NO (default), the field definition of sort key field in the SORT AREA declarative must be relative to the start of the field definition in the RELEASE statement.

If SRTADJ=YES, the field definition of area sort key fields must be the true offset relative to the start of the area. With SRTADJ=YES, you do not have to use PTRs or extra EQU statements to define the sort keys for the SORT declarative. Check with the person who installed VISION:Report to find out what option is the default.

Examples

SRTADJ=NO (default):

```
EQU       S-AREA             WST501-600
EQU       S-AREA-KEY1        WST501-502
EQU       S-AREA-KEY2        WST505-510
EQU       FAKE-KEY1          WST1-2
EQU       FAKE-KEY2          WST5-10

SORT AREA  RL100  ON   FAKE-KEY1   FAKE-KEY2

MOVE ...  TO S-AREA-KEY1               /*  Build your record
MOVE ...  TO S-AREA-KEY2

RELEASE   S-AREA  TO SORT
```

SRTADJ=YES:

The EQU statements for FAKE-KEY1 and FAKE-KEY2 are not required, so you can replace the SORT AREA statement above with:

```
SORT AREA  RL100  ON   S-AREA-KEY1  S-AREA-KEY2
```

Sort Area (V) and Multiple SORT Executions

An example of sorting variable length records and multiple executions of the SORT are shown below. The assumption can be made that a large parts inventory file is in sequence by product and part number.

The request is made for a listing from this file, in part name sequence, within product code (that is, file is already in sequence on the major control field). The SORT is invoked for each product code upon recognizing a change in the product code field.

```
        SORT AREA (V) RL160 ON INF35-49          /* SORT control
        ATEND 190                                /* At INF EOF, process last prod
        MOVE SPACES TO WST201-205                /* Clear prod control break field
    100 GET                                      /* Read an INF record
    110 IF INF5-9 IS EQ TO WST201-205
             MOVE INF1-2-B TO VAL203-204-B       /* Place record length in VAL
             RELEASE INF1 TO SORT                /* Record passed to SORT
             GOTO 100.
        IF WST201-205 IS BLANK
             MOVE INF5-9 TO WST201-205
             GOTO 110.
    *
    190 MOVE SPACES TO WST201-205                /* Clear prod control break field
        MOVE SPACES TO VAL200                    /* Clear SORT EOF indicator
    200 RETURN SORTED INTO WST1                  /* Record returned from SORT
        IF VAL196 IS EQ TO C'E'                  /* Test INF EOF
            IF VAL200 IS EQ TO C'E'              /*  Test SORT EOF
            GO TO EOJ.                           /* EQ, go to EOJ
        IF VAL200 IS EQ TO C'E'                  /* Test for SORT EOF
            GOTO 110.                            /* EQ, go process next prod
        MOVE WST35-49 TO PRT1
        MOVE WST5-9 TO PRT21
        MOVE....ETC
        PRINT                                    /* PRINT the record
        GOTO 200                                 /* Go return next record
        END
```

To clarify, the following code is correct:

```
        SORT AREA (F) RL525 ON PTA1-5
        SET PTA WST251
    100 GET INF ATEND 900
         .
         .
          Build record for sorting, starting at WST251.
         .
         .
        RELEASE PTA1 TO SORT
          GOTO 100
    0900 RETURN SORTED INTO WST251 ATEND 1000
```

Statement 900 could use OFA1 instead of WST251.

SORT FILE

```
SORT FILE {INA-INZ|DET} ON flddef [(D)] [flddef [(D)]...]
```

Term	Description
SORT FILE	Sorts one of your input files before you see the data. When you issue a GET or GET DET, you receive a sorted record.
INA-INZ DET	Enter the file name to be sorted (INA-INZ, DET).
ON	Required noise word.
flddef	Enter the equated data name or field definition of the field on which you desire the major sort sequence. If you define more than one sort field, each subsequent field is a more minor sort sequence than the last (for example, the most major sort sequence is the first field, then the second).
(D)	If the previous field is to be sorted into descending sequence, enter (D). Otherwise, enter the equated data name or field definition of the field you desire to be sorted next. The default is ascending sequence.

Examples

```
SORT FILE INF ON INF21-29 INF161-164-P (D) INF30
SORT FILE DET ON DET1-l2 DET16-B DET14-15-B
SORT FILE INC ON EMP-NAME                          /* EQU data name
SORT FILE IND ON GROSS (D)                         /* EQU data name
```

The SORT statement has the following limitations:

- All operands must be in one statement.
- Sum of all SORT fields may not exceed 256 bytes.
- The work area for the sort control statements is 512 bytes. As many fields may be specified as will fit in this area.
- File sorts must either be fixed or variable record types.
- The MEDIA (columns 4-7) of the SORT file specified cannot be TAPU or TAPV.

The following examples demonstrate how to specify the sorting of a VSAM file by Plant (major) and Department (minor), and how to print and accumulate totals for the file.

Be sure to include JCL for SORT work files. For MVS, include a SYSOUT DD statement, and possibly SORTLIB if this library is not in the concatenated list. Some JCL examples are shown below, with the additional JCL statements in bold print.

VSE JCL Example

```
// JOB QJSORT                                          SORT FILE
// DLBL SORTWK1,'SORTWK1',0
// EXTENT SYS003,volser,                               as examples
// ASSGN SYS003,DISK,VOL=volser,SHR
// DLBL filename,'your.VISION.lib'                     Phase Library
// EXTENT   ,volser
// LIBDEF  PHASE,SEARCH=(lib.sublib)
// DLBL   INF,'your.VSAM',,VSAM
// EXEC  QUKBJOB,SIZE=512K
INFKSDS    0080                                       /* VSE I/O parameter
  ... VISION:Report statements as shown below
/*
/&
```

MVS JCL Example

```
//QJSORT  JOB  (800-0000,0000),'SORT FILE'
//STEP1   EXEC  PGM=QUIKJOB,REGION=512K
//STEPLIB  DD  DISP=SHR,DSN=your.VISION.loadlib
//SYSOUT   DD  SYSOUT=*
//SORTLIB  DD  DSN=SYS1.SORTLIB,DISP=SHR              if needed
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DISP=SHR,DSN=your.VSAM                 INF
//SYSIN    DD  *
  ... VISION:Report statements as shown below
/*
//
```

VISION:Report Example

```
OPTION SEQCHK=NO
EQU PLANT INF1-2
EQU DEPT  INF4-5
*
SORT FILE INF ON PLANT DEPT          /* Define SORT fields
*
BREAK 1 DEPT ...                     /* Define control-break fields
BREAKS 2 PLANT ...
*
100 GET                              /* Sorted file is read
*
CHECKBREAKS                          /* Check control-break fields
MOVE PLANT TO PRT1
MOVE DEPT TO PRT8
MOVE ...
PRINT                                /* PRINT
ACCUM ...                            /* ACCUM field
GOTO 100
END
```

ADDITIONAL SORT OPTIONS

In addition to the SORT options that can be specified in the OPTION statement described in the section Sorting, there are several parameters that can also be specified on the SORT statement itself.

These specifications are all in the format of XXVVVVVVVV, where:

XX – is an option

VVVVVVVV – 1-8 byte value for the option above, unless specified otherwise.

ES – Estimated number of records. This overrides the OPTION SRTERCD=. This must be numeric.

Example: ES2400 – estimated number is 2400.

(OS/390 only)

GV – Overrides the VSE installation default sort GVSIZE=value for GETVIS sorting. There is no equivalent VISION:Report OPTION parameter for this. This must be numeric.

Example: GV123456 -- allocate 123,456 bytes of GETVIS storage.

(VSE only)

PG – SORT program name (executable module). This overrides the OPTION SORTPGM=.

Example: PGCASORT – invoke a program named "CASORT" to execute the SORT. If you have several different sort programs at your site, you can use a different one for each SORT. As an example, you could rename the IBM sort program name to DFSORT, and the CA sort program name to CASORT. You could sort multiple files, each with a possibly different sort program.

PR – sort message print option. This overrides OPTION SORTPRT=YES/NO. The VISION:Report SORTPRT=YES generates a sort PRINT=ALL, while SORTPRT=NO generates a sort PRINT=CRITICAL. Select NONE, ALL or CRITICAL.

Example: PRALL – print all messages.

RT – sort message routing. This overrides OPTION SORTRTE=.

Example: RTLOG – route sort messages to the LOG.

ST – storage value specified in number of bytes. This overrides the VISION:Report OPTION SORTSIZ=. This must be numeric.

Example: ST123456 – allocate 123,456 bytes of storage.

WF – number of work files. This overrides the VISION:Report OPTION SORTWRK=. This must be numeric.

Example: WF4 – allocates 4 work files for the SORT.

WN – name of work files. This must be 4 bytes long and requires the VISION:Report OPTION SRTWKN=YES. This overrides the default work file names.

Example: WNCUBS – the work file names become CUBSWKn

Following are some sample VISION:Report SORT statements:

SORT FILE INF (V) PGCASORT ST234500 PRCRITICAL WF3 WNCUBS ON ....

SORT AREA RL25 ST50000 PRALL GV200000 PGDFSORT ON ....

SUB

```
SUB {flddef1|C'xxx'|P'nnn'|X'...'} FROM flddef2
```

Term	Description
SUB	Subtracts the first value from the second value. The result is placed in the second value (flddef2).
flddef1 C'xxx' P'nnn' X'...'	Defines the flddef or literal with a limit of 19 digits EBCDIC or a packed field of 10 bytes (19 digits and sign). Binary fields 1-8 bytes long may be specified in either flddef. Fields and literals in any data format may be subtracted from any field in any other data format. Automatic data conversion is performed in all cases.
FROM	Required noise word.
flddef2	Defines the area/field name that is to be subtracted from flddef1. This also contains the result of the subtraction. The rules for flddef2 are the same as flddef1.

Examples

```
SUB INF1-3 FROM WST6-9          /* Subtract EBCDIC from EBCDIC
SUB INF7-9-P FR OFA10-19-P      /* Subtract packed from packed
SUB WST1-2-B FR WST11-16-B      /* Subtract 2 byte binary from 6 byte binary
SUB INF8-10-P FR PUN1-10        /* Subtract packed from EBCDIC
SUB INF47-51-B FR OFA21-26-P    /* Subtract 5 byte binary from packed
SUB C'1' FROM WST5-7            /* Subtract a literal from EBCDIC
SUB C'123' FR OFA7-19-P         /* Subtract a literal from packed
SUB C'4' FR OFX1-8-B            /* Subtract a literal from 8 byte binary
```

SUB treats fields with leading or all blanks (EBCDIC format) as zeros in the blank positions.

TABLSORT

```
TABLSORT {TSA1|flddef} C'entry-length' C'key-start' C'key-length' {C'A'|C'D'}
{C'C'|C'P'}
```

Term	Description
TABLSORT	This is a subroutine that sorts VISION:Report tables in place in memory according to user-defined sort specifications. TABLSORT is executed through coding a statement in the following format:
TSA, flddef	TSA1 passes to the routine the address of byte 1 of the first table entry. It is recommended that you code SET TSA INITIAL ahead of the CALL TABLSORT to ensure that TSA is pointing correctly at entry to TABLSORT. VISION:Report users may also specify sorting to begin at any VISION:Report area.
C'entry-length'	A 2-digit number equal to the length of one table entry. If each entry consists of 3-byte argument and a 5-byte function, then code C'08' for the entry length.
C'key-start'	A 2-digit number equal to the starting position of the sort key within the table entry. If you want to sort on the function in the 3/5 table, then code C'04' for the first byte of the 5-byte function.
C'key-length'	A 2-digit number equal to the length in bytes of the sort key. If sorting on the function on the 3/5 table, code C'05' for the length of the 5-byte function.
C'A', C'D'	If the table is to be sorted in ascending sequence on the specified key field, code C'A'. For a descending sequence sort, code C'D'.
C'C', C'P'	If the sort key field is other than packed decimal format, code C'C'. If the sort key field is packed decimal, code C'P'.

TABLSORT recustomizes itself (according to the user specifications) each time it is entered. You may, therefore, sort the table or data to several different sequences within one VISION:Report.

Sorting the table on other than a table argument sort key loses the binary table lookup integrity needed to execute an IF...ONTABLE. You must resort the table back to the argument ascending sequence before attempting IF...ONTABLE.

For an example of the use of the TABLSORT, see the section Examples in Chapter 4.

TABLSORT requires a high-value entry at the end of the table or data stream being sorted of at least one high-value (X'FF'). Therefore, you can sort account numbers, dates, amounts in examples like DET, INA-INZ, PRT and WST.

The first position of each entry may not be X'FF' and there must be one X'FF' at the end of the entries or groups to be sorted.

Example

Assume you want to print five position part numbers with one space in between, five per print line, and you want to sequence them lowest to highest.

Move the part numbers to the print area followed by one space. After moving the last one, move a high-value (X'FF') to the position in which the sixth part number would begin, and call TABLSORT or use the SORT fields statement to sort them, then print them.

```
PRT1            Print Work Area Before Sort

12345Ø12341Ø12349Ø88765Ø00123ØF
                                F

CALL TABLSORT PRT1 C'06' C'01' C'05' C'A' C'C'
```

Then print work area after sort

```
00123Ø12341Ø12345Ø12349Ø88765ØF
                                F
```

This technique works for transactions of, for example, account and amount, part number and price, and names, which can be in any sequence in the input data, but you can resequence before printing or processing.

If more than 2 digits are required for starting position and/or length fields, use the SORT fields statement, or use TABLSOR2 in place of TABLSORT.

Note: TABLSOR2 has the same format as TABLSORT, except that an 8-digit number is required for:

- Table entry-length — length of each table entry.
- Key-start — starting position of the sort key.
- Key-length — length in bytes of the sort key.

TABLSPEC

```
TABLSPEC max-entries arg-start arg-length [P] [funct-start funct-length] [LIST]
```

Term	Description
TABLSPEC	Defines the format of the variable data and the format of the loaded table. Only one TABLSPEC statement is allowed per VISION:Report program. TABLESPEC lets you load variables from complex IF statements into a data table. You can then reference the variables with the IF...ONTABLE statement.The TABLSPEC statement is free form. Each parameter can be any number of digits up to a maximum of 15, as long as it is terminated by at least one space. The next parameter then follows. While the parameters are considered free form, each parameter must follow in the sequence as specified in the fixed-format (for example, arg-length cannot be specified before arg-start).
max-entries	Maximum number of table entries. For any one execution of this job, you can have fewer than the maximum entries loaded. For example, if you specify 10, you can have from 1 to 9 entries. **Note:** The max-entries becomes high-values, not max-entries plus one. If your maximum entries are 50, you should add one more (51) for the last entry, which is filled with high-values to signify the end of the table.
arg-start	Leftmost position in the table statements of the argument field.
arg-length	Length in bytes of the argument field on the table. This cannot exceed 80 bytes when used in conjunction with IF...ONTABLE statements. If a packed argument is specified, the argument is packed directly from the table statement into the table. Therefore, the EBCDIC format argument in the table statements must be of the correct length to pack appropriately. Thus, if a 3-byte packed argument is specified then a 5-digit EBCDIC number must be prepared for each table statement starting in the statement column specified in TABLSPEC columns 19-20, if the fixed format of the TABLSPEC statement is used.
P	Format of the argument field. Leave blank if the argument is EBCDIC. Enter P for a packed format argument.

Term	Description
funct-start	Leftmost position in the table statements of the function field. If there is no function, this field may be left blank. If this operand is coded, funct-length must also be coded.
funct-length	Length of the function. All functions must be EBCDIC.
LIST	Enter LIST to cause table entries to be listed after they have been sorted.

Free Format

```
        TABLSPEC  11000 0900       5  P
0001..05...10...15...20...25...30...35...40...45...50...55
```

In the above example, 11000 is the maximum number of table entries, 0900 is the leftmost position in the table statements of the argument field, 5 is the length in bytes of the argument field on the table and the format is packed. The parameters were deliberately spaced apart from each other, with more than one space in between on most parameters, just to demonstrate the new TABLSPEC format. The above example used 0900, with leading zeros, though it is not necessary.

Fixed Format

The fixed-format, although not as flexible or required, is shown below for your convenience. The sequence of its contents are identical to the free format.

```
        TABLSPEC 0900 01 05
0001..05...10...15...20...25...30...35...40...45...50...55
```

In the above example, 0900 is the maximum number of table entries, 01 is the leftmost position in the table statements of the argument field, and 05 is the length in bytes of the argument field on the table. The format, normally in position 25, is blank, signifying EBCDIC. (It would have been filled with a P if packed.)

Column	Contents
1-3	Sequence numbers
5-12	TABLSPEC
14-17	Maximum number of table entries. For any one execution of this job, fewer than the maximum entries may be loaded.
19-20	Leftmost position in the table statements of the argument field (leading zeros are required).

Column	Contents
22-23	Length in bytes of the argument field on the table (leading zeros are required). This cannot exceed 80 bytes when used in conjunction with IF...ONTABLE statements. If a packed argument is specified, the argument is packed directly from the table statement into the table. Therefore, the EBCDIC format argument in the table statements must be of the correct length to pack appropriately. If the fixed format of the TABLSPEC statement is used, and if a 3-byte packed argument is specified, then a 5-digit EBCDIC number must be prepared for each table statement starting in the statement column specified in TABLSPEC columns 19-20.
25	Format of the argument field. Leave blank if the argument is EBCDIC. Enter P for a packed format argument.
27-28	Leftmost position in the table statements of the function field (leading zeros are required). If there is no function, this field may be left blank. If the funct-start operand is coded, funct-length must also be coded.
30-31	Length of the function (leading zeros are required). All functions must be EBCDIC.
33-36	Enter LIST to cause table entries to be listed after they have been sorted.

User Data Tables

To validate data or make changes to files you might code:

```
IF INF1-3 IS EQ TO C'TRN'
```

This example would validate a transaction code. However, this is not practical where there are 20 or 30 or more valid transaction codes.

Similarly, for a small number of changes to a file you might use the following code to change the contents of one part number field. Once again, it is not practical if there are many changes to be made.

```
IF INF15-21 IS EQ TO C'1234567'
    MOVE C'5467321' TO INF15-21.
```

To assist you in handling this type of situation, a user table loading feature is available. In the second example above, the following technique might be used:

```
IF INF15-21 IS ONTABLE
    MOVE FUN1-7 TO INF15-21.
```

Prepare statements, one statement per part number change, with the old part number in statement columns 1-7 and the new part number in columns 8-14. These statements would follow immediately behind the END statement.

A free form TABLESPEC statement would contain the following:

- Sequence number
- TABLSPEC
- Number of part number changes
- 01 (table statement column where argument starts)
- 07 (argument size)
- Blank (If argument were packed, a P would be entered)
- 08 (table statement column where function starts)
- 07 (function size)
- LIST (list sorted table entries)

All VSE memory available in the partition in excess of the basic VISION:Report requirements is available for table loading, called programs, and input/output areas and work areas. Table space is acquired before I/O space is acquired.

All MVS input output areas, I/O work areas, table space, and called programs are acquired through GETMAIN and/or LOAD MVS functions.

You need not be concerned with excessive run-time for larger tables, as the loaded table is searched with a binary table lookup. This technique searches the largest table in relatively few compares (that is, 14 compares on an 800 entry table).

To use a binary table lookup, the table must be in argument sequence. An internal sort is performed on the table after it has been loaded. When loading larger tables (over 1000 entries), the sort time is reduced significantly if, when loaded, the table is in sequence by argument.

The table is always sorted into key sequence if loaded by VISION:Report, unless the SORTABL option (see OPTION) specifies that the table is not to be sorted. If you request that the table not be sorted, then any IF...ONTABLE statement causes a serial search of the table starting at the first entry in the table. Also, note that if a search for the key fails and SORTABL=NO, TBH points to the next available position in the table in which a new entry may be loaded. This feature is very useful when building a table dynamically.

```
OPTION SORTABL=NO
005 TABLSPEC 0100 01 10   11 05
010 GET DET ATEND 100                  /* Get table record
020 IF DET1-10 IS ONTABLE              /* Look for argument
030     ADD C'1' TO TBH11-15           /* Found it-update counter
040     GOTO 010.
050 MOVE DET1-10 TO TBH1-10            /* TBH point to next available entry
060 MOVE C'00001' TO TBH11-15
070 GOTO 010
100 ....finish processing
```

In this example, after the table has been loaded, you could use the TABLSORT routine to sort the table into argument sequences. Even though the table has been sorted into argument sequence, a serial search is performed because SORTABL=NO.

A table can be loaded by one of two methods:

- Automatically

```
010 TABLSPEC 0100 01 07   08 07
020 GET                                /* Get detail record
030 IF INF1-7 IS ONTABLE               /* Lookup order # in table
040     MOVE TBH8-14 TO INF1-7.        /* Replace order # with new #
050 MOVE INF1-80 TO OFA1               /* Move input record to output
060 WRITE OFA                          /* Write output record
070 GOTO 020
999 END
11111111010101
33333333030303
66666666060606
00000011000000
```

In this example, the statements that follow the END statement are the user-defined table statements. VISION:Report loads these statements and sorts them into argument sequence unless the SORTABLE option is NO (see OPTION).

If the maximum number of table entries is 20 or less, a serial search is performed whenever an IF...ONTABLE statement is executed. If the maximum number of table entries is greater than 20, a binary search is performed whenever the IF...ONTABLE statement is executed, unless the SORTABL OPTION is NO, in which case a serial search is always performed.

- User Loading

```
TABLSPEC 0101 01 07   08 07                /* Max is really 100 entries
020 SET TSA INITIAL
025 MOVE C'000' TO WST1-3                  /* Initialize table entry count
030 GET DET ATEND 100                      /* Get table statement
040 MOVE DET1-14 TO TSA1-14                /* Move table entry to table
045 SET TSA UP 14                          /* Point to next table slot
050 ADD C'1' TO WST1-3                     /* Update # table entries
060 IF WST1-3 IS GT C'100'                 /* Check for table overflow
070     MOVE C'TABLE OVERFLOW' TO PRT1     /* Move error message
080     PRINT                              /* Print error message
090     GOTO EOJ.
095 GOTO 030
100 SET TSA INITIAL
110 CALL TABLSORT TSA1 C'14' C'01' C'07' C'A' C'C'    /* Sort table
120 GET INF ATEND EOJ                      /* Get input record
130 IF INF1-7 IS ONTABLE                   /* Lookup order # in table
140     MOVE TBH8-14 TO INF1-7.            /* Replace order # with new #
150 MOVE INF1-80 TO OFA1                   /* Move input record to output
160 WRITE OFA                              /* Write output record
170 GOTO 120
180 END
```

In the example above, for VSE, file I/O statements for INF and DET are required, as well as a "/*" (EOF) statement is also required after the VISION:Report statement "180 END".

The statements 020 through 095 are examples of statements that are used to load the table. Statements 100 and 110 sort the table into argument sequence (refer to TABLSORT for a discussion of this subroutine). When the IF...ONTABLE is executed, VISION:Report performs a binary search to find a match (table maximum is more than 20 entries) unless the SORTABL OPTION was NO. That OPTION causes a serial search to be performed starting at the first entry in the table.

Warning: If the following conditions are both true, you must ensure that the table is in argument sequence before the first IF...ONTABLE is executed. Failure to sort the table destroys the integrity of the binary search and results in false hits or no hits.

MAXIMUM TABLE ENTRIES IS GREATER THAN 20.

SORTABL=YES (installation default).

Advanced Techniques for Referencing Tables

Two advanced methods of referencing user tables are available.

- The ability to modify table entries as part of the true processing of an IF...ONTABLE statement.
- The ability to address each table entry through indexing (or subscripting) the table from the first entry to the last.

These advanced features are explained in detail below.

Referencing Hit Entries Following an IF...ONTABLE

The table lookup routine stores the address of the table argument that matches your search argument in an IF...ONTABLE statement. The table entry pointed at by the stored address has been assigned the area name TBH for table hit. By citing TBH as the area part of a field definition, you may reference any part of the table entry.

As an example, consider that you have loaded a table consisting of entries with a three-character argument followed by a five-character function consisting of EBCDIC zeros. The objective is to read a file and obtain a count of how many times each table argument occurs in the file. The following sequence illustrates count accumulation.

```
001 ATEND 200
.   .
.   .
010 GET
020 IF INF3-5 IS ONTABLE
030  ADD C'1' TO TBH4-8.            /* Give the length/data type within table
040 GOTO 010
.  .
.  .
```

When end of file is reached, the program transfers to statement 200. At this point, each table function contains a count of how many times its associated argument was contained in positions 3-5 of an input record.

Although modification of table arguments is allowed, it is not recommended. The ascending sequence of table arguments could be lost; if so, the integrity of binary search technique used by VISION:Report can no longer be ensured.

TBH may be referenced the same as any other VISION:Report area. The next successful table lookup causes TBH to point to a new entry. On a no-hit, TBH and FUN still reference the last successful IF. You must provide for this condition.

Indexing Through the Table

The ability to modify table entries creates a requirement that you be able to retrieve the modified entries. To demonstrate this, the following program shows the user loading the table and initializing parts of the table, usage of table start address (TSA) and the SET TSA statement, the IF ... ONTABLE condition, retrieval of the modified entries, and the previously introduced table hit (TBH).

Example

After loading the table, the TSA is set to the beginning with the SET TSA INITIAL statement; this is shown throughout this program. At sequence 010 through 019, the State table is loaded: 2 bytes for the state number, 22 bytes for its corresponding function, with the last 5 bytes used for a packed counter. Each table entry is 30 bytes long. Note that a check is made to make sure that the maximum number of entries, 55 is not exceeded (although in reality there are only 52 entries). Also note that high-values are moved to the last table entry. These are good practices to ensure that you do not go beyond the maximum number of table entries.

Statements 030 through 099 read an input record, containing a state number in the first two positions. A table lookup is automatically done by the IF INF1-2 IS ONTABLE statement; if a match is found (table hit), then the counter for that table entry is incremented and moved (along with the full-spelling of the state name) to the print area. The first two positions of the input are moved to print area and printed.

When the input file reaches end of file, statements 120 through 160 loop through all the table entries, printing only those that had any input activities (those with a counter that is non-zero). Note that a check is made for high-values for each table entry. When that table entry is reached, you know you are at the end of all pertinent table entries.

```
EQU TBL-COUNT    WST1-2-P   ZEROS
    TABLSPEC 0056 01 02   03 28           /* State number & name
    SET TSA INITIAL                       /* Ensure you are at beginning
010  GET DET ATEND 20                     /* Build table
     ADD C'1' TO TBL-COUNT
     IF TBL-COUNT GT P'55'                /* Allow a little for growth
        MOVE C'STATE TABLE OVERFLOW' TO PRT1
        PRINT
        ABEND 1234.
     MOVE DET1-25  TO TSA1                /* Move state number, name
     MOVE ZEROS    TO TSA26-30-P          /* Initialize counters
     SET TSA UP 30                        /* Increment to next table entry
019  GOTO 010                             /* Loop until table built
     SET TSA INITIAL                      /* Reset to beginning
030  GET INF  ATEND 100                   /* Read input records
     IF  INF1-2  IS ONTABLE               /* Got a hit on state?
         MOVE TBH3-25 TO PRT3             /* Also, show spelling
         ADD C'1'  TO TBH26-30-P.         /* Increment number of 'hits'
     MOVE INF1-2     TO  PRT1
     PRINT
099  GOTO 030                             /* Loop until input exhausted
100  SET TSA INITIAL                      /* Reset to beginning
120  IF TSA1-2  IS HIVALUES               /* Are we at the end?
         GOTO EOJ.                        /* Yes
     IF TSA26-30-P  EQ ZEROS              /* Any hits?
         GOTO 150.                        /* Do not print entries with zeros
     MOVE TSA29-30-P  TO PRT1             /* Truncate it
     MOVE C' FOR:'    TO PRT6
     MOVE TSA3-25     TO PRT11            /* Show states w/ hits
     PRINT
150  SET TSA UP 30                        /* Increment to next table entry
160  GOTO 120                             /* Loop till done, then EOJ
9999END
```

TCLOSE (VSAM ONLY)

```
TCLOSE {INF|INA-INZ|DET|OFA-OFZ}
```

Term	Description
TCLOSE	Complete outstanding I/O operations and update catalog, but does not disconnect the program from the data, formats the last CA in the file to ensure that all of the data that has been loaded is accessible. Writes SMF records if you are using SMF.
INA-INZ DET OFA-OFZ	This can be any VISION:Report input and/or output file name. INF is the default.

Example

```
GET INF                          /* Get a record
MOVE C'NEWINFO' TO INF10         /* Change the record
REWRITE INF                      /* Rewrite
TCLOSE INF                       /* Commit to Disk
```

TITLE/TITLE2/TITLE n

```
TITLE  [n] 'title information [reserved words]'
TITLE2 'title information [reserved words]'
```

Term	Description
TITLE TITLE2 TITLE n	Specifies the title contents without regard to print line positions.
n	Title line number. This can be a number from 1 to 6 (default is 1).

title information	Enclose in single quotation marks a character string for a report title. (The length cannot exceed the length of the PRTSIZE option.) All data between the enclosing quotation marks is considered title information, so leading, embedded, and trailing blanks may be specified. If you want a single quotation mark, code two consecutive quotation marks.
	
	TITLE, TITLE n, and TITLE2 replace HDA through HDF. HDC through HDF areas and functions are unchanged, except when the REPORT statement is used. In this case, REPORT controls the HDC through HDF areas. You can control the format of TITLE by submitting multiple TITLE lines whose contents are put together from left to right in the sequence that they are input. If the length of the composed TITLE line is less than PRTSIZE, then that TITLE is centered by VISION:Report.
	
	`TITLE 'Example of first title'`
	`TITLE 4 'Example of fourth title line'`
	`TITLE 'This is George''s test'`

Term	Description
reserved words	As with the HDR statement, you may code VISION:Report supplied reserved word constants. These are replaced with the appropriate value during execution. (VISION:Report supplies some useful reserved words ($datanames$), see HDR for reserved words $....$.) Page number and date are included in your report title when space allows. Additionally, you can specify valid equated data names or field definitions enclosed in dollar signs (reserved header names) to place user data fields in the header areas each time page headers are printed. The reserved words ($datanames$) are expanded or compressed as needed to correspond to the data field length. This feature is intended to simplify those cases where sections of a report are identified by file data placed in the page titles. This data should be moved from the record input area to a WST work area so that it can be accessed at end of file time.

When the CHECKBREAKS statement is used, the $dataname$ is substituted under the control of the BREAK spacing options and the BREAK level using the following scheme:

- Before totals line is printed:
  - SB=E and level=1. Move $dataname$ to headers after headers are printed.
  - SB=E and level=final. Move spaces to headers.
  - SB=E and other levels. Do not move $dataname$ to headers.
- After totals line is printed:
  - SB=E and level=1. Move $dataname$ to headers.
  - SB=E and next higher level=final. Move spaces to headers.
  - SB=E and other levels. Do not move $dataname$ to headers.
- For all other BREAK spacing and level combinations:
  - Move $dataname$ to headers.

Example

Here is a report in sequence by plant number that requires a page break between plants with the plant number appearing in the title.

```
TITLE 'LIST OF EXEMPT EMPLOYEES AT PLANT $WST112-114$'
```

or

```
TITLE 'LIST OF EXEMPT EMPLOYEES AT PLANT $PLANT-NR$'
```

If the first two plants in the file are 300 and 500, the first two section titles would read:

LIST OF EXEMPT EMPLOYEES AT PLANT 300
LIST OF EXEMPT EMPLOYEES AT PLANT 500

TRACE

```
TRACE {ALL|LAST50|OFF}
```

The TRACE statement is declarative, not executable. Therefore, it may not be the destination of a GOTO or PERFORM. Further, it defines what part of the program is to be traced, not when it is to be traced.

Term	Description
TRACE	Tracks the processing flow of a program. It also assists in troubleshooting a problem. (See the section Troubleshooting and Memory Requirements in Chapter 5 for more information concerning troubleshooting.)
ALL	Causes each statement to be posted and printed whenever the count specified by TRACECT has been reached (see OPTION) or a PRINT, PRINTCHAR, PRINTHEX, or DOHEADERS statement is executed. Up to 10 trace entries can be printed on a single line. When the trace line is printed, it consists of the internal VISION:Report sequence number (the number in parentheses that is printed to the left of each VISION:Report statement) and the four-character code for the statement. Trace lines are interspersed with any printing that is produced by the program.
LAST50	Causes a memory table of VISION:Report internal sequence numbers, executed after the trace was requested or activated, to be created and posted. No trace printing occurs on the printer unless the program aborts due to a program check. Up to and including the last 50 statement numbers executed are printed on the printer if a program check occurs. The last statement executed is identified. The last statement executed normally is the statement causing the program check, if the TRACE LAST50 statement was requested as the first statement of the program.
OFF	Turns off the trace routine. No further tracing occurs until a TRACE ALL or TRACE LAST50 statement is encountered.

A TRACE record is created for each statement executed in a VISION:Report program. Each record consists of the statement sequence number generated in a program listing and 3- or 4-byte abbreviation of the statement name.

If you specify TRACECT=1 and TRACE ALL, the address of the statement in memory is also listed in the trace record. Relations in compound IF statements are individually traceable. The first relation is identified by the statement sequence number and the remaining relations are specified by a number followed by an asterisk, and refer to the ordinal position of the relation within the statement.

Example

The following causes the entire program to be traced and the statements and their mnemonics to be listed on the printer as they are executed. Execution time will be increased.

```
    OPTION TRACECT=1
010 TRACE ALL
020 GET
030 ADD....
040 ADD....
050 MULT....
060 GO TO 020
```

The following causes the entire program to be traced and the last 50 statements executed placed in an internal memory trace table. No printing of trace statements occurs unless a program check occurs.

```
010 TRACE LAST50
020 GET
030 ADD....
nnn GO TO 020
```

The following traces statements 040 through 070 independent of the data.

```
OPTION TRACECT=1
010 GET
020 IF INF10-12 IS GT C'052'
030      TRACE ALL
040      MOVE.....
050      GO TO 010.
060 ADD...
070 TRACE OFF
080 GO TO 010
```

The following causes 020, 030, and 040 to be traced on the printer. No tracing occurs after statement 040.

```
OPTION TRACECT=1
010 TRACE ALL
020 GET
030 ADD...
040 MULT....
050 TRACE OFF
nnn
900 END
```

```
(22)       IF (
(23)        (
(24)        ( FLDB TMM FLDC AND                              /*  1
(25)          FLDC = FLDC  OR                                /*  2
(26)          FLDH INCLUDES FLDI FLDJ REVERSE ) OR            /*  3
(27)        ( FLDE = FLDE  OR                                /*  4
(28)          FLDF = FLDF ) AND                              /*  5
(29)        ( FLDE = FLDE  OR                                /*  6
(30)          FLDF = FLDF ) AND                              /*  7
(31)          FLDG = FLDG                                    /*  8
(32)          ) AND FLDC = FLDC                               /*  9
(33)          )
(34)          GOTO 999.

STMT    22 IF       00032050
STMT   3 * IF       00032090
STMT   9 * IF       00032154
STMT    34 GOTO     00032170
```

Recommendation:

TRACE LAST50 can be placed in all new programs when testing and debugging programs. This facilitates debugging in case of program checks and does not produce any extra paper, for example, unless the program aborts due to a program check. TRACE ALL and TRACE LAST50 cause 6 bytes of memory per statement extra to be used.

See the section Program Check Routine in Chapter 5 for a description of the other debugging aids that automatically occur on a program check, regardless of whether or not TRACE is being used.

Warning: The use of the TRACE statement causes the VISION:Report generated program to require approximately two thirds more CPU time to execute. It is not recommended that TRACE be left in for production jobs or jobs that have a large volume of data to process. For these types of jobs, use the SAMPLE and LIMITREADS statements along with one of the TRACE statements to debug the program before the production job is run.

TRAN

```
TRAN flddef1 WITH flddef2 { flddef3 | X'...' }
```

Term	Description
TRAN	Translates a data field using a 256-byte substitution list. The value of each byte in the source is added to the list address and the byte at the resulting address is substituted for the original byte value in the source.
flddef1	Source data to be translated.
WITH	Required noise word.
flddef2	A 256-byte list used to provide the substituted byte values.
flddef3 X'...'	Optional. A 2- or a 4-byte binary field indicating the number of bytes to be translated. If this operand is omitted, then the length of the source field is used.

Example

```
EQU FILLER       WST0
EQU FLDA         (20)  C'BLUEJAYS EAT PEANUTS'
EQU FLDB        (256)
EQU FLDB
EQU FILLER       (16)  X'FFFEFDFCFBFAF9F8F7F6F5F4F3F2F1F0'
EQU FILLER       (16)  X'EFEEEDECEBEAE9E8E7E6E5E4E3E2E1E0'
EQU FILLER       (16)  X'DFDEDDDCDBDAD9D8D7D6D5D4D3D2D1D0'
EQU FILLER       (16)  X'CFCECDCCCBCAC9C8C7C6C5C4C3C2C1C0'
EQU FILLER       (16)  X'BFBEBDBCBBBAB9B8B7B6B5B4B3B2B1B0'
EQU FILLER       (16)  X'AFAEADACABAAA9A8A7A6A5A4A3A2A1A0'
EQU FILLER       (16)  X'9F9E9D9C9B9A99989796959493929190'
EQU FILLER       (16)  X'8F8E8D8C8B8A89888786858483828180'
EQU FILLER       (16)  X'7F7E7D7C7B7A79787776757473727170'
EQU FILLER       (16)  X'6F6E6D6C6B6A69686766656463626160'
EQU FILLER       (16)  X'5F5E5D5C5B5A59585756555453525150'
EQU FILLER       (16)  X'4F4E4D4C4B4A49484746454443424140'
EQU FILLER       (16)  X'3F3E3D3C3B3A39383736353433323130'
EQU FILLER       (16)  X'2F2E2D2C2B2A29282726252423222120'
EQU FILLER       (16)  X'1F1E1D1C1B1A19181716151413121110'
EQU FILLER       (16)  X'0F0E0D0C0B0A09080706050403020100'
    TRAN FLDA WITH FLDB
    PRINTHEX FLDA

                 WST1-20
                         32132311B331B2332111
                         DCBAEE7DFAECF8AEABCD
                        01..05...10...15...20
```

TRNT

```
TRNT flddef1 WITH flddef2 { flddef3 | X'...' }
```

Term	Description
TRNT	Translates and tests a data field using a 256-byte substitution list. The value of each byte in the source is added to the list address and the function byte at the resulting address is inspected for a value of zero. The source field remains unchanged. The bytes in the source field are examined one by one from left to right until a non-zero function byte is encountered or all the bytes in the source field have been examined. The results of the operation are returned in VAL224-B, VAL225-228-B and PTR. If no non-zero function bytes were encountered, then VAL224-B and VAL225-228-B are both zero and PTR is unchanged. If a non-zero function byte is encountered, then it is returned in VAL224-B, the number of bytes scanned is returned in VAL225-228-B and PTR points to the byte in the source that resulted in the non-zero function byte.
flddef1	Source data to be tested.
WITH	Required noise word.
flddef2	A 256-byte list used to provide the substituted byte values.
flddef3 X'...'	Optional. A 2- or a 4-byte binary field indicating the number of bytes to be translated and tested. If this operand is omitted, then the length of the source field is used.

Example

```
* FOLLOWING EXAMPLE SHOULD STOP ON THE ‘E’ OF BLUEJAYS
EQU FILLER       WST0
EQU FLDA         (20)  C'BLUEJAYS EAT PEANUTS'
EQU FLDB        (256)
EQU FLDB             /*  0 1 2 3 4 5 6 7 8 9 A B C D E F
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 0
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 1
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 2
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 3
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 4
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 5
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 6
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 7
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 8
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 9
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* A
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* B
EQU FILLER       (16)  X'00000000000100000000000000000000'  /* C
EQU FILLER       (16)  X'00010000000200000000000000000000'  /* D
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* E
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* F
```

```
TRNT FLDA WITH FLDB
    PRINTHEX FLDA
    PRINTHEX PTR1
    PRINTHEX VAL224-228
```

```
          WST1-20 BLUEJAYS EAT PEANUTS
                  CDECDCEE4CCE4DCCDEEE
                  23451182051307515432
                 01..05...10...15...20
           PTR1-1 E
                  C
                  5
                 01
        VAL224-228
                  00000
                  10004
                  01..05
```

WHEN

```
WHEN flddef1 {INCLUDES|OMITS} {flddef2|C'xxx'|X'xxx'|SPACE|BLANK|NONSPACE|NONBLANK}
[flddef3] [REVERSE] {AND|OR}
```

Note: You can code this statement using IF in place of WHEN. You can also insert the parameters of this statement in a compound IF statement.

Term	Description
WHEN	This statement performs a left-to-right scan (or optionally a right-to-left scan) of the field specified. If the byte string specified in flddef2 is present within the scanned field, a user addressable pointer is set (special area PTR) to the leftmost (rightmost if reversed) position of the found match. The number of bytes scanned is returned in VAL225-228-B. Processing continues with the next statement. If a match is not found, processing is transferred to the statement following the period. (See Rule K in General Rules.)
flddef1	Define the field to be scanned in the standard field definition format. Variable field lengths may be any binary value in a 2- or 4-byte field (see condition operand).
INCLUDES OMITS	Determines whether or not to INCLUDES or OMITS based on conditions.

Term	Description
condition	Enter the search argument that establishes a true condition. flddef2 Define the search argument in standard field format. If the argument length is greater than the variable scan field length, processing is transferred to the next sentence. C'xxxx' Maximum length for character literals is 40 bytes. X'xxxx' Maximum length for hexadecimal literals is 36 bytes. SPACE Same as literal X'40' one byte long. BLANK Same as literal X'40' one byte long. NONSPACE Any byte except X'40' one byte long. NONBLANK Any byte except X'40' one byte long.
flddef3	Required for variable length scan fields. Flddef must be a binary field of 2- or 4-bytes long that contains a value indicating the length of data to be scanned. **Warning**: If the length of this operand is in error and causes VISION:Report to go beyond its area, region or partition, program checks can occur. If the length in this operand is smaller than the length implied in the condition, processing is transferred to the statement following the period.
REVERSE	Changes the normal direction of the scan from left-to-right to right-to-left. REVERSE causes the scan to begin at the rightmost position of the field and continue until the leftmost position is reached, or a hit is found.

Term	Description

AND, OR	States the relationship of multiple WHEN and/or IF statements.
	
	IF statements may be interspersed with WHEN statements.
	
	AND is the default. AND requires that all statements must meet all conditions before the true statements are executed. If any of the WHEN statements fail, a fall through occurs to the next sentence. The following example illustrates the AND logic.
	
	`140 WHEN INF1-80 INCLUDES C'OHIO' AND`
	`150 IF BILL-AMOUNT IS GT ZERO`
	`160     WHEN INF1-80 INCLUDES C'CURRENT'`
	`170     MOVE INF1-80 TO OFA1-80`
	`180     MOVE OFA1-80 TO PRT1`
	`190     WRITE OFA`
	`200     PRINT.`
	`210 GO TO 10`
	
	If input positions INF1-80 contains OHIO and BILL-AMOUNT is greater than zero, and INF1-80 contains CURRENT, the balance of the sentence is executed. If INF1-80 does not contain both OHIO and CURRENT or if BILL-AMOUNT is not greater than zero, the relationship is false and control is transferred to statement 210, the beginning of the next sentence.
	
	VAL225-228-B, after a successful hit, contains the number of bytes scanned over until the leftmost position (rightmost position when using the REVERSE option) of the successful hit was found. If an unsuccessful hit or false condition is the result, VAL225-228-B contains the number of bytes scanned from the last successful WHEN hit, or binary zeros if no previous hit was successful.

Term	Description

OR	The use of the OR connective is valid only when any number of WHEN...OR statements are followed by a WHEN statement. This is necessary to indicate the end of the OR logic. Any WHEN statements which follow it are not part of the preceding OR logic. A found false condition causes the program to fall through to the next statement which must be an IF or WHEN. If the word OR is not included, a found true condition causes fall through to the next statement while a found false condition causes transfer to the next sentence.
	
	`040 IF INF1-7 IS NOT EQUAL TO C'NEW3380' OR`
	`050 WHEN INF1-80 INCLUDES C'3380' OR`
	`060 WHEN INF1-80 INCLUDES C'3375' OR`
	`070 WHEN INF1-80 INCLUDES C'3350'`
	`080 MOVE C'3380' TO PRT1-4`
	`090 MOVE INF1-80 TO PRT10`
	`100 PRINT.`
	`110 MOVE INF1-80 TO OFA1-80`
	
	If input positions INF1-80 contains one of 3380, 3375, 3350, or if INF1-7 does not equal NEW3380, then processing transfers to statement 080 and the balance of the sentence is executed. If INF1-7 does equal NEW3380 or INF1-80 does not contain 3380, 3375, 3350, the relationship is false and control is transferred to statement 110, the beginning of the next sentence (the statement following the period in 100 PRINT).

Examples

```
                                                Scan    Field   Search
    Statement                                   From    Thru    Argument

WHEN INF1-80 INCLUDES C'CURRENT'                INF1    INF80   CURRENT

WHEN INF1-80 INCLUDES C'X''180'''               INF1    INF80   X'180'

WHEN DET1-80 INCLUDES SPACES                    DET1    DET80   (SPACES)

MOVE C'45' TO INF1-4-B
WHEN INF1-80 INCLUDES C'VARIABLE' INF1-4-B      INF1    INF45   VARIABLE

WHEN INF1-80 INCLUDES C'LAST' REVERSE           INF80   INF1    LAST

MOVE C'45' TO INF1-4-B
WHEN INF5-80 INCLUDES NONBLANK INF1-4-B         INF5    INF49   NOT''

MOVE C'45' TO INF1-4-B
WHEN SAV12 INCLUDES X'FF' INF3-4-B REVERSE      SAV56   SAV12   X'FF'
```

WRITE

```
WRITE {OFA-OFZ} [ONERROR seq-no]
```

Term	Description
WRITE	Writes the contents of the area specified to the appropriate output file.
OFA-OFZ	Write the contents of OFx area to the OFx file (for example, write the contents of OFA area to OFA file).
ONERROR seq-no	Valid for VSAM files only. The ONERROR operand causes VISION:Report to automatically transfer to the statement with the specified sequence number when a VSAM error occurs. This allows you to examine the error codes RC/EC and take appropriate action.

For VSAM, the length of the record must be specified in the 2-byte or 4-byte length field preceding the I/O area. See the techniques shown in the section VSAM Recommendations in Chapter 2.

The following lists the errors returned when the ONERROR operand is not coded. These errors apply to KSDS files only.

Error	Error Word VAL253-255	RC/EC VAL246-248
DUPLICATE RECORD	'DUP'	X'0808'
RECORD OUT OF SEQUENCE	'SEQ'	X'080C'

In VSE, after execution of the WRITE statement, the contents of the area OFA-OFZ as applicable are no longer available to you except in the case of a VSAM file. A form of PUT LOCATE processing is used to support the WRITE statement. When the WRITE is issued, the pointer (to the appropriate place in the output area where the data record is located or placed) is incremented to the next output record position and/or a new block.

In MVS, after execution of the WRITE statement, the contents of the area OFA-OFZ are still available to you. MOVE mode of processing is used to support the WRITE statement. The file pointer always points to a work area that is the same size as the largest record in the output file.

Example

```
010 GET INF                          /* Get an input record
020 MOVE INF1-100 TO OFA1-100        /* Move input to your output file A
030 MOVE OFA1-100 TO OFB1-100        /* Move output area A to output area B
040 MOVE OFB1-100 TO OFC1-100        /* Move output area B to output area C
050 WRITE OFA                        /* Write OFA
060 WRITE OFB                        /* Write OFB
070 WRITE OFC                        /* Write OFC
080 GOTO 010                         /* Go get next record
```

For VSE users, this example shows reading a file and making three copies of it using output files A, B, and C. Notice that OFA1-100 is moved to OFB1-100 and OFB1-100 is moved to OFC1-100. This can be done until either a WRITE OFA, WRITE OFB or WRITE OFC statement is issued. If a WRITE OFA statement is inserted as statement 021, the OFA area pointed to by statement 030 would not contain the records that were read from the input file. It might be a previous record (residue from a previously written block) or it might be garbage.

While the above is correct, it is not recommended that output areas be used as the source of data for other purposes.

XOR (Logical XOR)

```
XOR flddef1 WITH {flddef2 | C'xxx' | P'nnn' | X'...' | ZERO | BLANK | SPACE |
HIVALUE | LOVALUE } { flddef3 | X'...' }
```

Term	Description
XOR	Performs the logical XOR (Exclusive OR) of a data field. The contents of a bit position in the source are set to one if the corresponding bit positions in the operands are unlike; otherwise the bit is set to zero. The resulting condition code is returned in VAL223-B. If any bits in the source operand are one following completion of the operation, then the condition code is x'01'; otherwise, it is x'00'. In the case of a ZERO figcon, if flddef1 is character, then a character zero x'F0' is used, otherwise, a binary zero x'00' is used.
flddef1	Source data. This field may be changed because of the operation.
WITH	Required noise word.
flddef2 C'xxx' P'nnn' X'...' ZERO BLANK SPACE HIVALUE LOVALUE	The second operand. This field is not changed.
flddef3 X'...'	Optional. A 2- or a 4-byte binary field indicating the number of bytes to be XORd. If this operand is omitted, then the length of the source field is used.

Example

```
EQU FILLER       WST0
EQU FLDA         (6)   X'304050607080'
EQU FLDB         (6)   X'303030303030'
    XOR FLDA WITH FLDB
    PRINTHEX FLDA
    PRINTHEX VAL223

                WST1-6   -&
                       07654B
                       000000
                      01..05.
            VAL223-223
                       0
                       1
                      01
```

Chapter

4

Examples

Examples

This chapter contains VISION:Report examples for VSE and MVS. The associated JCL is shown in the first few examples. Since the JCL is fairly standard, sample JCL is only shown where it is significantly different. For MVS, these examples are in the SAMPLIB PDS. For VSE they are located in the VISION:Report library created at installation. In either case, the members are named SAMP followed by a 2-digit number signifying the example number.

Some of the examples also use optional material, which is described in greater detail in Chapter 6.

Example	Type
Example 1	Load / Copy Tape to Disk
Example 2	Copy Card File to Two Tape Files, One Blocked and Standard Label, One Unblocked and Unlabeled
Example 3	Variable Disk Input, Variable Tape Output
Example 4	Variable Record Output, Table Lookup, Indexing, PRINTHEX Variable, PERFORM, HDR, OPTION STMTEND
Example 5	Create AR VSAM KSDS File Using Native VSAM from Sequential Disk, SORT File in Building VSAM Key
Example 6	Concatenate Two Undefined Record Files into One Undefined Output File
Example 7	File Maintenance or File Matching
Example 8	Table Lookup, Range Checking, Negative Field Testing
Example 9	Multiple Tables, Alphanumeric Checking
Example 10	Table Data for Repricing
Example 11	Accumulating Amounts in a Table, Print at EOJ

Example	Type
Example 12	Dynamically Create and Sort a Table, Accumulate, and Print at EOJ
Example 13	Table Load, TABLSORT, Print Various Sequences, Multiple HDR, and Various OPTION Parameter Overrides
Example 14	Native VSAM Using GET, QUIKIPDS/QUIKINCL, REPORT, SORT AREA with RELEASE/RETURN, DISPLAY, CALL to QUIKDATE
Example 15	Additional Working Storage and QUIKVSAM, Using OPTION, POINT, GET-UPD, ERASE, and MOVE with Quotes
Example 16	TABLSPEC, Indexing, Table ACCUM, BREAK, Summary Output to Disk
Example 17	ACCUM Counts, Amounts Using CTR-NO, BREAK, CHECKBREAKS, QUIKIPDS, LIMITREADS, PAGETOTALS, REPORT, SORT, and Numerous IF Statements While Validating
Example 18	ACCUM Using CTR, BREAK, and CHECKBREAKS
Example 19	ACCUM Using CTA-CTC, BREAK, CHECKBREAKS, and Summary Output, PUNCH
Example 20	ACCUM, BREAK, and CHECKBREAKS
Example 21	ACCUM, BREAK, and CHECKBREAKS with Total Time Calculations, Multiple HDR
Example 22	Amortization Schedule, Calculations, No Input/Output Files, LINECOUNT, Arithmetic Operations, Multiple HDR, PERFORM
Example 23	Match Records of a Transaction File Against a Master File and Create a New Master file
Example 24	Print Report with OMIT, SORT AREA, SRTADJ, and RPTSPCE
Example 25	Print Report Summary
Example 26	SET PCC, MOVE VARIABLE LENGTH, EQU with Literals, Negative Numbers, WHEN and WHEN/REVERSE, QUIKVSAM with Read-Upd and Update
Example 27	Native VSAM using GET, SET PTA, PRINTHEX
Example 28	Native VSAM (RRDS) using GET and SETGENKEY
Example 29	Native VSAM (RRDS) using WRITE
Example 30	Native VSAM (KSDS, RRDS, ESDS) Using Random Access, READ, ADDRECORD, REWRITE, DELETE

Example	Type
Example 31	Native VSAM using GET, QUIKIPDS, WHEN, WHEN/REVERSE, IF...NUMERIC, IF...ALPHA, Negative IF, Multiple HDR with $names$
Example 32	Native VSAM (ESDS) with Alternate Index, Using OPEN/CLOSE, GET, READ, SETGENKEY, REWRITE, SET PTA
Example 33	Native Variable Length VSAM (KSDS, ESDS) Using OPEN/CLOSE, GET, WRITE, SET PTA, READ, SETGENKEY, ONERROR
Example 34	QUIKVSAM (KSDS) with Alternate Index, Using OPTION, OPEN/CLOSE, LOAD, READ, GET-UPD, READ-UPD, ADD, GET, POINT, UPDATE, ERASE
Example 35	Troubleshooting Problems
Example 36	Mixture of CALLing QUIKVSAM and native VSAM, field names greater than 14 characters, and $PAGE$ reserved word
Example 37	Nested IF, IF with parentheses, IF/ELSE/ENDIF, and bit manipulation instructions such as AND, OR, XOR, TRAN, TRNT
Examples 38A and 38B	IF Statement with Test Under Mask Operands

JCL Examples

In all the examples that follow, the JCL is fairly standardized. The first few examples show JCL for VSE and MVS, and their relationships to the VISION:Report statements. In addition, usage of PROCs, as distributed in the Release tape, are also shown. As the examples progress, the JCL is no longer shown, as it would be similar to the first few examples. In some cases, the VISION:Report statements show the I/O statements with an asterisk (*) in front; the asterisks should be removed if the operating system is VSE.

VSE JCL Example

```
// JOB REPORT Card input, Tape output
// DLBL filename,'your.VISION.lib'                          (1)
// EXTENT   ,volser                                         (2)
// LIBDEF  PHASE,SEARCH=(lib.sublib)                        (3)
// LIBDEF SOURCE,SEARCH=(lib.sublib)                        (4)
```

Statement (4) is required if the QUIKINCL exit is used. See Example 14.

Use the following statements to standardize your JCL, where QJTEST has been catalogued as a PROC with statements 1-4 as shown in the previous example.

```
// DLBL  QJLIB,'your.VISION.proclib'    PROC Library
// EXTENT   ,volser
// LIBDEF  PROC,SEARCH=QJLIB.PROC
// EXEC PROC=QJTEST

// TLBL   OFA,'dataset.name'        Output file OFA on I/O stmt
// ASSGN  SYS010,580                Assign SYS nr. to phy.device as required
// EXEC  QUKBJOB,SIZE=512K
INFCARD                             /*  VSE I/O parameter stmt as necessary
OFATAPE800000800SSYS010             /*  VSE I/O parameter stmt as necessary
...  VISION:Report statements as required
9999END                             (Ensure the END statement is here)
.Card input
.  goes here
/*                                  /* VSE EOD data delimiter
/& VSE EOJ
```

A PROC, QJTEST, contained on the Distribution tape, assists in providing general guidelines for the minimum JCL. It provides for library assignments, allowing for future patch libraries, but ASSGN, TLBL, DLBL for input and output files should be supplied according to the needs of your installation.

MVS JCL Example

```
//REPORT   JOB  (800-0000,0000),'CARD TO DISK'
//STEP1   EXEC  PGM=QUIKJOB,REGION=512K             (1)
//STEPLIB  DD  DISP=SHR,DSN=your.VISION.loadlib     (2)
//SYSPRINT DD  SYSOUT=*                             (3)
//QUIKIPDS DD  DISP=SHR,DSN=your.VISION.source      (4)*
//SYSUT2   DD  DISP=(,CATLG),SPACE=    etc.
//             UNIT=SYSDA  Output dataset, if required, for OFA
//SYSIN    DD  *
   VISION:Report statements as required
9999END                            /* Ensure the END statement is here
//SYSUT1   DD  *
.Card input
.  goes here
/*
//
```

*Statement is required if the QUIKIPDS exit is used. See Example 14.

A PROC, QJTEST, contained on the Release tape, assists in providing general guidelines for the minimum JCL, as shown in statements 1-4 above. It can also provide for STEPLIB statements, allowing for future patch libraries; the necessary DD statements for input and output files will still need to be supplied according to the needs of your installation. If the PROC is used, you may use a statement similar to the following to override any JCL statements within the PROC:

```
//RUNIT  EXEC  QJTEST
//QJ.SYSUT1  DD    DSN=etc.              For INF file
//QJ.SYSUT2  DD    DSN=etc.              For OFA file
```

Within the VISION:Report programs, there can be "commented-out" VSE I/O statements (starts with an '*' in position 1). For VSE users, delete the '*' in position 1 of the VSE I/O parameter statement, shifting all data on that statement one byte to the left. In the first few examples, the true format of these VSE I/O parameter statements is shown with the appropriate JCL. After that, it appears in the VISION:Report program as a comment.

If you use a SORT in the VISION:Report program, SORT WORK JCL for devices/data sets is required for intermediate storage, as well as any PHASE library or LOADLIBs for the SORT program. Examples are:

VSE JCL with SORT Example:

```
// DLBL SORTWK1,'SORTWK1',0
// EXTENT SYS003,volser,etc.
// ASSGN SYS003,DISK,VOL=volser,SHR
```

MVS JCL with SORT Example:

```
//SYSOUT    DD  SYSOUT=*
//SORTWK01  DD  UNIT=SYSDA,SPACE=(TRK,(5,5))
//SORTWK02  DD  UNIT=SYSDA,SPACE=(TRK,(5,5))
```

Example 1

Load/Copy Tape to Disk

This example shows how to read data records from tape and restore them to disk. The records are written to disk in a blocked, fixed-format, with the record size and block size the same as the tape. This example shows all the JCL and VISION:Report statements necessary for completing this job in either a VSE or MVS environment (see JCL Examples).

The Accounts Receivable file is copied from the Release tape of VISION:Report and restored to disk.

VSE JCL Example

```
// JOB TPTODISK             Tape to Disk - Restore AR file from tape
// DLBL filename,'your.VISION.lib'
// EXTENT   ,volser
// LIBDEF  PHASE,SEARCH=(lib.sublib)
// TLBL   INF,'input.arfile'
// ASSGN  SYS010,580        Assign SYS nr. to phy.device as required
// DLBL   OFA,'ARFILE'
// EXTENT SYS006,volser,1,0,start track,number tracks
// ASSGN  SYS006,DISK,VOL=SER=volser,SHR
// EXEC  QUKBJOB
INFTAPE52800352SSYS010       /* This is what the VSE I/O parameter
OFADISC52800352SSYS006       /* statements should look like
...  VISION:Report statements as shown below
/*
/&
```

MVS JCL Example

```
//TPTODISK JOB  (800-0000,0000),'TAPE TO DISK'
//STEP1   EXEC  PGM=QUIKJOB,REGION=512K
//STEPLIB  DD  DISP=SHR,DSN=your.VISION.loadlib
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DISP=OLD,DSN=input.arfile,etc.
//             UNIT=TAPE                              INF
//SYSUT2   DD  DISP=(,CATLG),SPACE=etc.
//             DSN=ARFILE,
//             DCB=(BLKSIZE=5280,LRECL=352,RECFM=FB),
//             UNIT=SYSDA                             OFA
//SYSIN    DD  *
...  VISION:Report statements as shown below
/*
//
```

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP01:    COPY TAPE FILE TO DISK.  COPY THE ACCOUNTS    *
*            RECEIVABLE (AR) FILE FROM RELEASE             *
*            DISTRIBUTION TAPE TO SEQUENTIAL DISK.         *
*                                                          *
************************************************************
*INFTAPE52800352SSYS010           /* If VSE, remove * at position 1
*OFADISC52800352SSYS006           /* If VSE, remove * at position 1
010  GET
     MOVE  INF1-352      TO  OFA1
     WRITE OFA
     GOTO 010
9999 END
```

Example 2

Copy Card File to Two Tape Files, One Blocked and Standard Label, One Unblocked and Unlabeled

This example reads a card image file and creates two tape files from the same card image file. One tape file is blocked 16000 bytes (blocking factor is 200) and has standard labels; the other tape file is unblocked (blocking factor = 1) and is unlabeled. This example shows all JCL and VISION:Report statements necessary for completing this job in either a VSE or MVS environment.

The JCL demonstrates usage of the QJTEST PROC that was distributed with the Release tape and aids in standardizing your VISION:Report jobstreams. Check with the person that installed VISION:Report and verify that the PROC has been modified and is available. In the event that the PROC is not available, review JCL Examples for examples of JCL that does not use the QJTEST PROC.

VSE JCL Example

```
// JOB CDTOTAPE                    Card to 2 Tape files
// DLBL  QJLIB,'your.VISION.proclib'PROC Library
// EXTENT   ,volser
// LIBDEF  PROC,SEARCH=QJLIB.PROC
// EXEC PROC=QJTEST
// TLBL   OFB,'dataset.name1'
// ASSGN  SYS010,580               Assign SYS nr. to phy.device for OFB
// ASSGN  SYS011,581               Assign SYS nr. to phy.device for OFA
// EXEC  QUKBJOB
INFCARD                            /* This is what the VSE I/O parameter
OFBTAPE    0080SSYS010  BS=16000   /* statements should look like
OFATAPE00800080NSYS011             /* VSE I/O parameter statement
...  VISION:Report statements as shown below
.. Card input file here
..
/*
/&
```

MVS JCL Example

```
//CDTOTAPE JOB  (800-0000,0000),'CARD TO 2 TAPES'
//RUNIT  EXEC QJTEST   Demonstrate usage of PROC
//QJ.SYSUT2   DD  DISP=(,KEEP),LABEL=(,NL),          OFA
//               DSN=dataset.name2,
//               DCB=(BLKSIZE=80,RECFM=F),
//               UNIT=TAPE
//QJ.SYSUT3   DD  DISP=(,CATLG),LABEL=(1,SL),        OFB
//               DSN=dataset.name1,
//               DCB=(BLKSIZE=16000,LRECL=80,RECFM=FB),
//               UNIT=TAPE
//QJ.SYSIN    DD  *
...  VISION:Report statements as shown below
//QJ.SYSUT1   DD  *                                  INF
.. Card input file here
..
/*
//
```

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP02:    COPY CARDS TO 2 TAPES,                        *
*            ONE TAPE FILE IS BLOCKED, STANDARD LABEL(OFB).*
*            ONE TAPE FILE IS UNBLOCKED, UNLABELLED (OFA). *
*                                                          *
************************************************************
*INFCARD                          /* IF VSE, REMOVE * AT POSITION 1
*OFBTAPE    0080SSYS011  BS=16000  /* IF VSE, REMOVE * AT POSITION 1
*OFATAPE00800080NSYS010            /* IF VSE, REMOVE * AT POSITION 1

010  GET                         /* GET INPUT RECORD
     MOVE INF1-80  TO  PRT1      /* MOVE TO PRINT LINE
     PRINT                       /* ECHO INPUT ON PRINTER
     MOVE INF1-80  TO  OFA1      /* MOVE TO FIRST OUTPUT
     WRITE OFA                   /*    AND WRITE IT OUT
* WRITE OUT SECOND FILE
     MOVE INF1-80  TO  OFB1      /* MOVE TO SECOND OUTPUT
     WRITE OFB                   /*    AND WRITE IT OUT
     GOTO 010                    /* LOOP UNTIL DONE
99999END
01ALABAMA
02ALASKA
03ARIZONA
04ARKANSAS
05CALIFORNIA
06COLORADO
07CONNECTICUT
08DELAWARE
09DISTRICT OF COLUMBIA
10FLORIDA
11GEORGIA
12HAWAII
13IDAHO
14ILLINOIS
15INDIANA
16IOWA
17KANSAS
18KENTUCKY
19LOUISIANA
20MAINE
21MARYLAND
```

```
22MASSACHUSETTS
23MICHIGAN
24MINNESOTA
25MISSISSIPPI
26MISSOURI
27MONTANA
28NEBRASKA
29NEVADA
30NEW HAMPSHIRE
31NEW JERSEY
32NEW MEXICO
33NEW YORK
34NORTH CAROLINA
35NORTH DAKOTA
36OHIO
37OKLAHOMA
38OREGON
39PENNSYLVANIA
40PUERTO RICO
41RHODE ISLAND
42SOUTH CAROLINA
43SOUTH DAKOTA
44TENNESSEE
45TEXAS
46UTAH
47VERMONT
48VIRGINIA
49WASHINGTON
50WEST VIRGINIA
51WISCONSIN
52WYOMING
/*
```

Example 3

Variable Disk Input, Variable Tape Output

This example reads a variable-blocked file from a disk device and copies all records to tape output. In addition, it checks for the presence of the current-days (MMDDYY) located anywhere from position 11 to the end of each record.

On the presence of a matching date, extend the length of the output record by 20 bytes and move the date into the first 6 bytes of the extension.

Disk input specifications: maximum block size=15000, maximum record size=300, device=3380, and override the default ddname or file name to CPVDATA. Tape output specifications: maximum block size=15000, maximum record size=320, labels=standard.

VSE JCL Example

```
// JOB VARCOPY VARIABLE DISK IN, VARIABLE TAPE OUT
// DLBL filename,'your.VISION.lib'
// EXTENT   ,volser
// LIBDEF  PHASE,SEARCH=(lib.sublib)
// DLBL  CPVDATA,'input.dataset.name'INF (label override)
// EXTENT SYS006,volser
// ASSGN  SYS006,DISK,VOL=SER=volser,SHR
// ASSGN  SYS010,580           Assign SYS nr. to phy.device as required
// TLBL OFA,'dataset.name'     OFA
// EXEC  QUKBJOB
INFDISV    0300SSYS006 BS=15000,LBL=CPVDATA   /* VSE I/O parameter
OFATAPV    0320SSYS010 BS=15000               /* VSE I/O parameter
...  VISION:Report statements as shown below
/*
/&
```

MVS JCL Example

```
//VARCOPY  JOB  (800-0000,0000),'TAPE TO DISK'
//STEP1   EXEC  PGM=QUIKJOB,REGION=512K
//STEPLIB  DD  DISP=SHR,DSN=your.VISION.loadlib
//SYSPRINT DD  SYSOUT=*
//CPVDATA  DD  DISP=SHR,DSN=input.dataset.name      INF
//SYSUT2   DD  DISP=(,CATLG),
//             DSN=dataset.name,
//             DCB=(BLKSIZE=15000,LRECL=320,RECFM=FB),
//             UNIT=TAPE                          OFA
//SYSIN    DD  *
OPTION INFDD=CPVDATA              /* Override DDNAME (SYSUT1) for INF
...  VISION:Report statements as shown below
/*
//
```

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP03:    VARIABLE DISK INPUT,                          *
*            VARIABLE TAPE OUTPUT.                         *
*                                                          *
*            OVERRIDE STANDARD DDNAME FOR INF.             *
*                                                          *
************************************************************
*INFDISV    0300SSYS010 BS=15000,LBL=CPVDATA  /* VSE I/O PARAMETER
*OFATAPV    0320SSYS011 BS=15000   /* IF VSE, REMOVE * AT POSITION 1
     MOVE VAL5-6 TO WST1             /* INITIALIZE - MOVE MM TO WST1-2
     MOVE VAL8-9 TO WST3             /* MOVE DD TO WST3-4
     MOVE VAL11-12 TO WST5           /* MOVE YY TO WST5-6
100  GET                             /* READ A RECORD
     MOVE BLANKS TO OFA1-300         /* CLEAR OUTPUT RECORD BUFFER
     MOVE INF1 TO OFA1               /* MOVE RECORD TO O/P FILE
     SET PTA OFA1                    /* SET INDEX TO BEGIN AT OFA1
     SET PTA UP OFA1-2-B             /* INCREMENT TO END OF RECORD
     WHEN OFA11-300 INCLUDES WST1-6  /* SCAN OUPUT RECORD FOR DATE
     MOVE PTR1-6 TO PTA1             /* MOVE FOUND DATE TO RECORD END
     MOVE OFA1-2-B TO WST11-12-P     /* MOVE VAR RECORD LENGTH TO WST
     ADD C'20' TO WST11-12-P         /* ADD 20 TO RECORD LENGTH
     MOVE WST11-12-P TO OFA1-2-B.    /* MOVE NEW LENGTH BACK TO RECORD
     WRITE OFA                       /* WRITE OUTPUT RECORD
     GOTO 100                        /* GO READ NEXT RECORD
99999END
/*
/&
* $$ EOJ
```

Example 4

Variable Record Output, Table Lookup, Indexing, PRINTHEX Variable, PERFORM, HDR, OPTION STMTEND

This example illustrates several VISION:Report features, and the printed output displays how powerful and flexible a tool VISION:Report can be. Note the following:

- Blocked variable length records are created and written to the OFA file. Maximum block size is 250 bytes, and maximum record size is 100 bytes. The first record is 50 bytes long with each succeeding record being incremented in length by 3 bytes up to a maximum of 100 bytes. The variable record length is specified in positions 1-2 (in binary format) of the output record. The decimal equivalent of the record length is also placed in positions 11-13 in EBCDIC format.
- PTA-PTB indexing is employed together with IF...ONTABLE to use the decimal record-length digits in positions 11-13 as an argument in finding the corresponding English-spelled name of each respective digit and placing it into the variable output record.
- The table area is hex printed in the standard PRINTHEX fixed-length format (TSA0001-0084). The variable length records follow on the printer output, each being hex printed using the PRINTHEX variable option.

VISION:Report Statements

```
OPTION  STMTEND=80
************************************************************
*                                                          *
* SAMP04:    VARIABLE RECORD DISK OUTPUT.                  *
*            TABLE LOOKUP, INDEXING, PRINTHEX VARIABLE,    *
*            PERFORM/THRU, HDR OPTION STMTEND.             *
*                                                          *
************************************************************
*OFADISV02500100SSYS006            /* IF VSE, REMOVE * AT POSITION 1
    HDR 1A 1                DEMO INCLUDING - VARIABLE OUTPUT - TABLE LOOKUP
    HDR 1B INDEX POINTER-PRINTHEX VARIABLE DATE: $IPLDAT$ PAGE: $PG$
    LINECOUNT 60           /* SET PRT LINES TO 60.
    TABLSPEC 0012 01 01   02 06 LIST /* TABLE SPECS - MAX ENTRIES=12
                           /* ARGUMENT BEGINS AT 1, FOR LENGTH 2
    SET TSA INITIAL        /* FUNCTION BEGINS AT 3, FOR LENGTH 9
    MOVE ZEROES TO WST1-3  /* VARIABLE VALUE INITIALIZED TO 0
    PRINTHEX TSA1-84       /* PRINTHEX TABLE AREA
100 MOVE C'050' TO WST5-7  /* BASE LENGTH OF VARIABLE RECORDS
    ADD WST1-3 TO WST5-7   /* INCREMENT RL WITH VAR. VALUE.
    MOVE LOVALUE TO OFA1-4 /* PLC BINARY ZEROES IN OFA AREA FOR VAR RLXX.
    MOVE WST5-7 TO OFA1-2-B /* MOVE VALUE TO OUTPUT AREA FOR REC LENGTH.
    MOVE WST5-7 TO OFA11    /* ALSO PLACE VALUE IN REC IN EBCDIC FORMAT.
    SET PTA OFA1            /* INITIAL PTA TO FRONT OF REC.
    SET PTA UP WST5-7       /* BUMP PTA UP BY VAR. VALUE, END OF RECORD.
    SET PTA DOWN 10         /* BACK OFF 10 BYTES FOR RIGHT SIDE OF RECORD.
    SET PTB WST7            /* INITIAL PTB TO LOW ORDER VALUE FIELD.
    PERFORM 300 THRU 400    /* DO TABLE LOOKUP AND PLACE ALPHA-SPELLED
```

```
    PERFORM 300 THRU 400    /* NAMES OF THE DIGITS INTO OFA OUTPUT AREA.
    PERFORM 300 THRU 400    /* WHICH FORM THE VAR. VALUE & RECORD LENGTH.
    PRINTHEX OFA1 OFA1-2-B  /* PRINTHEX RECORD FOR VARIABLE LENGTH.
    WRITE OFA               /* WRITE THE OUTPUT RECORD.
    ADD C'3' TO WST1-3      /* INCREMENT THE VARIABLE VALUE BY 3.
    IF WST1-3 IS GT C'050'  /* COMP VAR VALUE TO 50, FOR GR-THAN COMPARE
    GOTO EOJ.               /* ON TRUE CONDITION, FORCE EOJ PROCESSING.
    GOTO 100                /* GO DO MORE OUTPUT PROCESSING.
300 IF PTB1 IS ONTABLE      /* DO TABLE LOOKUP FOR DIGIT LOCATED AT PTB1.
        MOVE TBH2-7 TO PTA1 /* ON HIT, PLACE FUNCTION NAME IN PTA AREA.
        GOTO 350.           /* GO AROUND NO-HIT PROCESSING.
    MOVE C'****' TO PTA1    /* NO-HIT ON TABLE, MOVE * TO PTA AREA.
350 SET PTB DOWN 1          /* DECREMENT PTB INDEX BY 1.
    SET PTA DOWN 10         /* DECREMENT PTA INDEX BY 10.
400 EXIT                    /* EXIT PERFORM PROCESSING HERE.
9999END                     /* END OF STATEMENTS, TABLE FOLLOWS.
0ZERO
1ONE
2TWO
3THREE
4FOUR
5FIVE
6SIX
7SEVEN
8EIGHT
9NINE
```

Example 5

Create AR VSAM KSDS File Using Native VSAM from Sequential Disk, Sort File in Building VSAM Key

This example illustrates how to load a VSAM KSDS file from the sequential accounts receivable file that was restored in Example 1. This example sorts the AR file by account code and account number, which is used to build the VSAM key. Prior to running the actual VISION:Report program, you will need to run an IDCAMS step, which would look similar to the following:

```
DELETE (ARFILE.VSAM) CLUSTER
SET MAXCC=0
DEFINE CLUSTER (NAME (ARFILE.VSAM)        -
     VOL (volser)                         -
     RECSZ(352 352)   KEY (9,210) )       -
     DATA ( NAME (ARFILE.VSAM.DATA)       -
     SPEED                                -
     TRK (3,1) FREESPACE (20,5) )         -
INDEX ( NAME (ARFILE.VSAM.INDEX) )
```

VSE JCL Example

```
// JOB VSAMLOAD                           Load VSAM KSDS from SEQ AR File
// DLBL  QJLIB,'your.VISION.proclib'      PROC Library
// EXTENT   ,volser
// LIBDEF  PROC,SEARCH=QJLIB.PROC
// EXEC PROC=QJTEST
// DLBL   OFA,'ARFILE.VSAM',,VSAM         OFA
// DLBL   INF,'ARFILE'                    INF
// EXTENT SYS006,volser
// ASSGN  SYS006,DISK,VOL=SER=volser,SHR
// EXEC  QUKBJOB
OFAKSDS    0352
INFDISC52800352SSYS006
...  VISION:Report statements as shown below
/*
/&
```

MVS JCL Example

```
//VSAMLOAD JOB  (800-0000,0000),'LOAD VSAM FROM DISK'
//RUNIT  EXEC QJTEST                          From PROC Lib
//QJ.SYSUT1   DD  DISP=SHR,DSN=ARFILE         INF
//QJ.SYSUT2   DD  DISP=SHR,DSN=ARFILE.VSAM    OFA
//QJ.SYSIN    DD  *
...  VISION:Report statements as shown below
/*
//
```

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP05:    LOAD ACCOUNTS RECEIVABLE (AR) VSAM KSDS FILE  *
*            USING NATIVE VSAM FROM SEQUENTIAL DISK        *
*            CREATED IN ARBUILDT.  SORT FILE FIRST         *
*            BY ACCOUNT CODE, THEN ACCOUNT NUMBER TO       *
*            BUILD VSAM KEY.                               *
*                                                          *
************************************************************
*INFDISC52800352SSYS010                  /* If VSE, remove * at position 1
*OFAKSDS    0352                         /* If VSE, remove * at position 1
     SORT FILE INF ON INF182-183  INF4-10

     OPEN OFA
     SET PTA OFA1
     SET PTA DOWN 2                      /* Point to rec len fld
     MOVE P'352'         TO  PTA1-2-B
     MOVE  SPACES        TO  OFA1-352
010  GET  INF  ATEND EOJ
     MOVE  INF1-210      TO  OFA1-210
* BUILD KEY WITH NEXT 2 INSTRUCTIONS
     MOVE  INF182-183    TO  OFA211-212   /* Account-code
     MOVE  INF4-10       TO  OFA213-219   /* Acct
     MOVE  INF220-352    TO  OFA220-352   /* Rest of it

     MOVE  OFA211-219    TO  PRT1
     MOVE  INF85-109     TO  PRT30
     PRINT
     WRITE OFA   ONERROR   200
     GOTO 010

200  MOVE  C'==== ERROR DURING LOADING'     TO PRT20
     MOVE  OFA211-219    TO  PRT1
     PRINT
     GOTO 010
9999 END
```

Example 6

Concatenate Two Undefined Record Files into One Undefined Output File

This example copies the contents of two undefined-length format tape files into one tape file.

Maximum record length for the input files is 3000 bytes and 3500 bytes, respectively. Maximum record length for the output file is 3500 bytes.

VSE JCL Example

```
// JOB MERGE                      Merge undefined tapes into one tape
// DLBL filename,'your.VISION.lib'
// EXTENT   ,volser
// LIBDEF  PHASE,SEARCH=(lib.sublib)
// TLBL  INF,'undef.in1'          INF
// TLBL  DET,'undef.in2'          DET
// TLBL  OFA,'undef.out'          OFA
// ASSGN  SYS010,580              Assign SYS nr. for  INF
// ASSGN  SYS011,581              Assign SYS nr. for  DET
// ASSGN  SYS012,582              Assign SYS nr. for  OFA
// EXEC  QUKBJOB
...  VISION:Report statements as shown below
/*
/&
```

MVS JCL Example

```
//MERGE    JOB  (800-0000,0000),'MERGE UNDEFINED TAPES'
//STEP1   EXEC  PGM=QUIKJOB,REGION=512K
//STEPLIB  DD  DISP=SHR,DSN=your.VISION.loadlib
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  DISP=SHR,DSN=undef.in1    INF
//SYSDET   DD  DISP=SHR,DSN=undef.in2    DET
//SYSUT2   DD  DISP=(,CATLG),
//             DSN=undef.out,
//             DCB=(BLKSIZE=3500,LRECL=3500,RECFM=U),
//             UNIT=TAPE                 OFA
//SYSIN    DD  *
...  VISION:Report statements as shown below
/*
//
```

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP06:    COPY THE CONTENTS OF 2 UNDEFINED-LENGTH       *
*            FORMAT TAPE FILES INTO ONE UNDEFINED TAPE     *
*            FILE.                                         *
*                                                          *
************************************************************
*INFTAPU30003000SSYS010      /* If VSE, remove * at position 1
*DETTAPU35003500SSYS011      /* If VSE, remove * at position 1
*OFATAPU35003500SSYS012      /* If VSE, remove * at position 1

010 GET                     /* Read data from file 1.
    IF INF1-10 IS HIVALUE   /* Test for EOF.
         GO TO 100.         /* Branch to 100 on EOF.
    MOVE INF1-3000 TO OFA1  /* Move max RL to output.
    WRITE OFA               /* Write output.
    GO TO 010               /* Go read more data-file 1.

100 GET DET                 /* Read data from file 2.
    IF DET1-10 IS HIVALUE   /* Test for EOF on file 2.
         GO TO EOJ.         /* Force EOJ on eof.
    MOVE DET1-3500 TO OFA1  /* Move max RL to output.
    WRITE OFA               /* Write output.
    GO TO 100               /* Go read more data-file 2.
9999END
```

VISION:Report returns the length of each undefined record to VAL-1-4-P on input files. On output files, you must place the length of the record to be written in VAL1-4-P. Since the above example copies the undefined files to output unchanged, the length is contained in VAL1-4-P, but is not referenced.

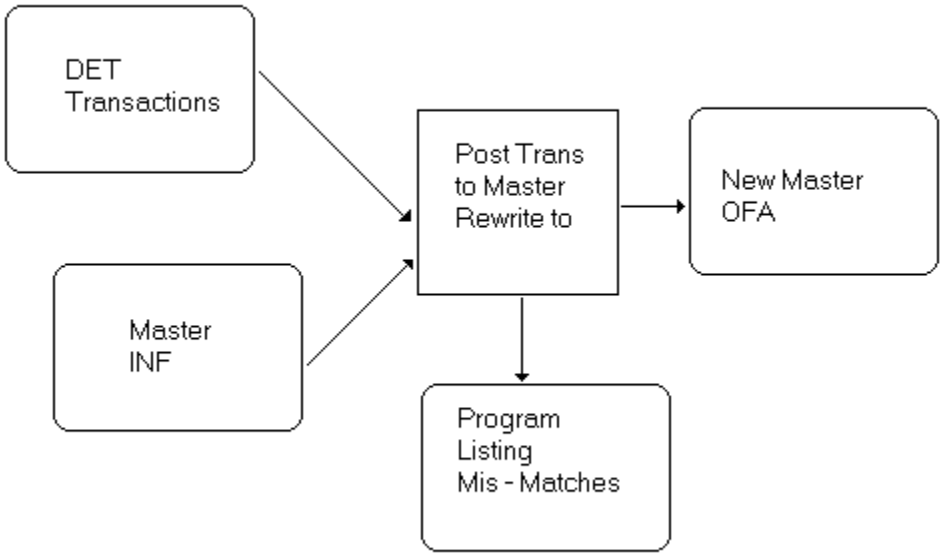
Example 7

File Maintenance or File Matching

This example shows you how to:

- Prepare a program which posts maintenance transactions to a master file. The files are 80-position records with the key in positions 1-4. Assume the master is a personnel or payroll type file.
- Allow for change or update type transactions; therefore, each transaction should match the master. Multiple matching transactions are allowable.
- Recreate or copy the master to a new area or disk extent. The master is on sequential disk. The transactions are on disk.

The flow of the problem looks like this:



VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP07:    MATCH MASTER AND TRANSACTION INPUT FILES      *
*            CREATE NEW MASTER FILE.                       *
*            ALLOWS FOR UPDATING OR CHANGING RECORDS,      *
*            AS WELL AS ADDING NEW RECORDS.                *
*                                                          *
************************************************************
*INFDISC16000080SSYS005           /* IF VSE, REMOVE * AT POSITION 1
*DETDISC32000080SSYS010           /* IF VSE, REMOVE * AT POSITION 1
*OFADISC16000080SSYS006           /* IF VSE, REMOVE * AT POSITION 1

EQU TRANS-KEY    DET1-9
EQU MSTR-KEY     INF1-9
EQU OLD-MASTER   INF1-80
EQU NEW-MASTER   OFA1-80

    GET INF                              /* READ A MASTER.
030 GET DET                              /* READ A TRANSACTION.
100 IF TRANS-KEY IS NOT EQ MSTR-KEY
        GOTO 160.
    IF MSTR-KEY EQ HIVALUE  AND          /* ENSURE BOTH NOT AT EOF
      IF  TRANS-KEY EQ HIVALUE
        GO TO EOJ.                       /* YES, GO TO EOJ.
    MOVE DET1-80 TO INF1-80              /* POST TRANS TO MASTER.
    GO TO 030                /* GO READ TRANS & RETURN TO MATCHING

160 IF TRANS-KEY IS GT MSTR-KEY          /* IF TRANS GT MASTER
        MOVE OLD-MASTER TO NEW-MASTER    /* MOVE MASTER TO OUTPUT.
        WRITE OFA                        /* WRITE IT OUT.
        GET INF                          /* GET THE NEXT MASTER.
        GO TO 100.                       /* GO TO THE MATCHING CODE.
200  /* ERROR - NO MATCHING KEY
    MOVE DET1-80 TO PRT1      /* ERROR SOMEWHERE
    PRINT
    GO TO 030                /* FETCH NEXT TRANSACTION.
9999END
```

Example 8

Table Lookup, Range Checking, Negative Field Testing

Given a transaction file containing 80-byte fixed-blocked records, this example completes the following edits:

- match the transaction code to a table entry
- verify that the dollar amount field is within the allowable low and high range limits of the matching table entry
- check the dollar amount field to ensure it is not negative

Any transactions not passing the above edits are printed along with an appropriate message identifying the test where the edit failed.

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP08:    EDIT TRANSACTION FILE.                        *
*            MATCH AGAINST TABLE ENTRY.                    *
*                                                          *
************************************************************
*INFTAPE16000080NSYS000     /* IF VSE, REMOVE * AT POSITION 1

EQU ARG-CODE      INF5-6
EQU $AMT          INF22-27
EQU LO-RANGE      FUN1-6
EQU HI-RANGE      FUN7-12
    TABLSPEC 0050 01 02   03 12 LIST
040 GET                                 /* READ A RECORD.
    IF ARG-CODE IS ONTABLE              /* DO TABLE LOOKUP.
       GOTO 100.                        /* IF ONTABLE, BR TO 100.
    MOVE C'INVALID TRANS. CODE' TO PRT45 /* ERROR MESSAGE.
075 MOVE INF1-40 TO PRT1                /* RECORD TO PRINT.
    PRINT                               /* PRINT ERROR RECORD.
    GOTO 040                            /* GO GET MORE RECORDS.
100 IF $AMT IS LT LO-RANGE OR           /* CK $AMT FOR LESS-THAN LO-RAN
       IF $AMT IS GT HI-RANGE           /* OR GREATER-THAN HI-RANGE.
       MOVE C'INVALID $AMOUNT' TO PRT45  /* ERROR, FILE/TABLE AMTS NOT =
       GOTO 075.                        /* GO PRINT RECORD.
    MOVE $AMT TO WST1-4-P               /* MOVE AMT TO PACKED FLD.
    IF WST1-4-P IS LT P'0'              /* CK FOR LESS THAN ZERO (NEGAT
       MOVE C'$ AMOUNT IS CREDIT' TO PRT45  /* IF SO, ERROR.
       GOTO 075.                        /* GO TO PRINT RECORD.
    MOVE C'** GOOD RECORD **' TO PRT45   /* DISPLAY GOOD RECORD MESSAGE
    MOVE INF1-40 TO PRT1                /* RECORD TO PRINT.
    PRINT                               /* PRINT ERROR RECORD.
    GOTO 040                            /* GO READ MORE RECORDS.
99999999END
```

```
01100000100000
02010000010000
M3001000001000
04000100000100
05000010000010
05000001000001
06200000200000
07020000020000
08002000002000
09000200000200
10000020000020
11000002000002
12300000300000
/*
```

Example 9

Multiple Tables, Alphanumeric Checking

Presume in an application the account number, department, and product codes require verification. This can be done by entering all such valid codes into a VISION:Report table and preceding each with a unique character. In this example, each account is preceded with an A, department with a D, and product with a P.

This example also verifies that a non-zero or non-blank product field has a fabrication code containing alphabetic characters and that the earnings and standard fields are numeric. This example prints all records not passing these edit checks along with the reason for failure.

VISION:Report Statements

```
*************************************************************
*                                                           *
* SAMP09:    VERIFY ACCOUNT NUMBER, DEPARTMENT AND PRODUCT  *
*            CODE AGAINST A TABLE.                          *
*            DO VARIOUS VALIDATION SUCH AS NUMERIC CHECKING *
*                                                           *
*************************************************************
OPTION LISTABL=YES
*INFTAPE16000080SSYS004           /* IF VSE, REMOVE * AT POSITION 1

EQU ACCT-NR  INF11-14                /* EQUATE
EQU DEPT-NR  INF15-17                /* DATA
EQU PROD-NR  INF18-23                /* NAMES
EQU FAB-CODE INF32-33                /* TO
EQU REG-EARN INF41-46                /* INF
EQU STD-EARN INF55-60                /* AREAS.
    HDR 1A 1 $IPLDAT$       DAILY TRANSACTIONS EDIT PAGE $PG$
    TABLSPEC 0600 01 07
    MOVE SPACES TO WST1-30          /* BLANK WORK-STORAGE AREA.
    MOVE C'A' TO WST1               /* PRECEDE EACH ACCT WITH 'A'.
    MOVE C'D' TO WST11              /* PRECEDE EACH DEPT WITH 'D'.
    MOVE C'P' TO WST21              /* PRECEDE EACH PROD WITH 'P'.

100 GET                             /* READ A RECORD.
    MOVE ACCT-NR TO WST2            /* MOVE ACCT BEHIND A.
    MOVE DEPT-NR TO WST12           /* MOVE DEPT BEHIND D.
    MOVE PROD-NR TO WST22           /* MOVE PROD BEHIND P.
    SET TSA INITIAL                 /* ENSURE IT'S CLEARED
    MOVE C'*** WST1-27'  TO PRT28
    MOVE WST1-27         TO PRT1
    PRINT
    IF WST1-7 IS ONTABLE            /* SEARCH TABLE FOR ACCT-NR.
       GOTO 200.                    /* FOUND, BR TO 200.
    MOVE C'ACCT-NR NOT ONTABLE' TO PRT55   /* NOT FOUND.
    GOTO 500                        /* PRINT RECORD AND ERROR CONDITION.
200 IF WST11-17 IS ONTABLE          /* SEARCH TABLE FOR DEPT-NR.
       GOTO 250.                    /* FOUND, BR TO 250.
    MOVE C'DEPT-NR NOT ONTABLE' TO PRT55 /* NOT FOUND.
    GOTO 500                        /* PRINT RECORD AND ERROR CONDITION.
250 IF WST21-27 IS ONTABLE          /* SEARCH TABLE FOR PROD-NR.
```

```
       GOTO 300.
    MOVE C'PROD-NR NOT ONTABLE' TO PRT55  /* NOT FOUND.<F255D>
    GOTO 500                      /* PRINT RECORD AND ERROR CONDITION.
300 IF PROD-NR IS BLANK OR        /* BYPASS FAB-CODE CHK IF BLANK.
       IF PROD-NR IS ZERO         /* OR ZERO PROD-NR.
       GOTO 400.
    IF FAB-CODE IS ALPHA          /* OK FOR ALPHA CHAR.
       GOTO 400.                  /* IF SO, GO TO 400.
    MOVE C'FAB-CODE NOT VALID' TO PRT55
    GOTO 500
400 IF REG-EARN IS NUMERIC        /* CHK FOR NUMERICS.
       GOTO 450.
    MOVE C'REG-EARN NOT NUMERIC' TO PRT55
    GOTO 500
450 IF STD-EARN IS NUMERIC        /* CHK FOR NUMERICS.
       GOTO 100.
    MOVE C'STD-EARN NOT NUMERIC' TO PRT55
500 MOVE INF1-50 TO PRT1          /* MOVE RECORD TO PRINT.
    PRINT DOUBLESPACED            /* PRINT IT.
    GOTO 100
9999END                           /* END QJ, TABLE FOLLOWS.
A1000
A1020
A1024
D001
D002
D005
D100
D104
D999
P000001
P000002
P000005
P100215
P100302
P999999
/*
```

Example 10

Table Data for Repricing

A master file used for pricing of invoices needs to be updated to incorporate new selling prices. The price increases are to be applied by commodity group varying from 1.0 percent to 12.5 percent of the current price.

In this example:

1. Place the approximately 75 commodity codes and their respective percentage of increases into a table.
2. Print only the items which change, showing the old and new selling price.
3. Write a new master file of changed and unchanged records.

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP10:    UPDATE MASTER FILE ON PRICING OF INVOICES.    *
*            USE TABLE.                                    *
*                                                          *
************************************************************
*INFTAPE20000200SSYS006           /* If VSE, remove * at position 1
*OFATAPE20000200SSYS005           /* If VSE, remove * at position 1

EQU SELL-PRICE INF51-55-P
EQU COMM-CODE  INF16-19
EQU ITEM-NR    INF2-8
    HDR 1A 1 $IPLDAT$          COMMODITY PRICE INCREASE       PAGE $PG$
    HDR 2A 0    ITEM       COMM         OLD PRICE      NEW PRICE      %
    TABLSPEC 0100 01 04   05 03 LIST

100 GET                          /* Get a record.
    IF COMM-CODE IS NOT ONTABLE  /* Search table for comm code.
         MOVE INF1-40 TO PRT1-40
         MOVE C'NOT ON TABLE' TO PRT45
         GOTO 270.               /* Not found; write out unchg record
    MOVE ITEM-NR TO PRT4
    MOVE COMM-CODE TO PRT16
    MOVE SELL-PRICE TO PRT25 2C
    MOVE TBH5-7 TO PRT58 3C
    MULT SELL-PRICE 2D BY TBH5-7 3D GIVING WST1-4-P 2DR   /* Calc. incr.
    ADD WST1-4-P TO SELL-PRICE   /* Add to old cost.
    MOVE SELL-PRICE TO PRT40 2C  /* Move new cost to print.
270 PRINT
    MOVE INF1-80  TO OFA1        /* Move input to output.
    WRITE OFA                    /* Write output.
    GOTO 100                     /* Go read more records.
9999END                          /* Statements end, table follows.
```

```
01CA005
02CA010
03CA015
04CA020
05CA025
06CA030
07CA035
08CA040
09CA045
10CA050
11CA055
12CA060
13CA065
14CA070
15CA075
16CA080
17CA085
18CA090
19CA095
20CA100
21CA105
22CA110
23CA115
24CA120
25CA125
26CB005
27CB010
28CB015
29CB020
30CB025
31CB030
32CB035
33CB040
34CB045
35CB050
36CB055
37CB060
38CB065
39CB070
40CB075
41CB080
42CB085
43CB090
44CB095
45CB100
46CB105
47CB110
48CB115
49CB120
50CB125
51CC005
52CC010
53CC015
54CC020
55CC025
56CC030
57CC035
58CC040
59CC045
60CC050
61CC055
62CC060
63CC065
64CC070
65CC075
```

```
66CC080
67CC085
68CC090
69CC095
70CC100
71CC105
72CC110
73CC115
74CC120
75CC125
/*
```

Example 11

Accumulating Amounts in a Table, Print at EOJ

Using a general ledger transaction file, accumulate and print dollar amounts for each account number matching a table entry.

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP11:    ACCUMULATE AMOUNTS IN A TABLE FROM A          *
*            GENERAL LEDGER TRANSACTION FILE.              *
*            PRINT DOLLAR AMOUNTS AT EOJ.                  *
*                                                          *
************************************************************
*INFTAPE40000500SSYS043            /* IF VSE, REMOVE * AT POSITION 1

    TABLSPEC 0200 01 04   05 08 LIST    /* MAX 20, ACCT=1-4,AMT=5-12.
    HDR 1A 1 $IPLDAT$      G/L ACCOUNT SUMMARY
    HDR 2A 0            ACCT              $AMOUNT

    ATEND 200           /* AT END-OF-FILE ON INF, BRANCH TO 200 FOR
    *                      DUMPING AND PRINTING OF ACCUMULATED TABLE
    *                      ENTRIES.  WE WILL FIRST LOOP THROUGH TABLE,
    *                      ZAPPING ZEROES INTO FUNCTION POSITIONS 5-12
    *                      SO WE CAN USE THEM AS COUNTERS.
    SET TSA INITIAL         /* SET INDEX TABLE POINTER AT START.
050 IF TSA1 IS HIVALUE      /* TEST FOR END OF TABLE (X'FF').
       GOTO 100.               /* EQ, BRANCH TO 100.
    MOVE ZEROES TO TSA5-12-P   /* MOVE ZEROES TO ACCT ENTRY FUNCTION.
    SET TSA UP 12           /* INCREMENT INDEX TO NEXT ENTRY.
    GOTO 050                /* GO TO START OF LOOP.

100 GET                         /* READ A G/L RECORD.
    IF INF5-8 IS ONTABLE        /* SEARCH A TABLE FOR HIT ON ACCT NR.
       ADD INF21-28-P TO TBH5-12-P.   /* FOUND, ADD $ AMT TO TABLE CTR
    GOTO 100                    /* GO READ NEXT RECORD.
    *
    *    EOF PROCESSING FOLLOWS
200 SET TSA INITIAL             /* POINT INDEX AT TABLE START.
210 IF TSA1 IS HIVALUE          /* TEST FOR END OF TABLE.
       GOTO EOJ.                /* EQ, GO FORCE EOJ.
    MOVE TSA1-4 TO PRT13        /* MOVE ACCT-NR TO PRINT.
    MOVE TSA5-12-P TO PRT20 2C  /* PRINT 2 DEC & COMMAS.
```

```
    PRINT DOUBLESPACED          /* PRINT TABLE ENTRY.
    SET TSA UP 12               /* INCREMENT INDEX TO NEXT TABLE ENTRY.
    GOTO 210                    /* GO PROCESS NEXT TABLE ENTRY.
9999END
A005
A010    €
A015
A020
A025    &
A030    -
A035    ø
A040    Ø
A045    °
A050
A055
A060    €
A065
A070
A075    &
A080    -
A085    ø
A090    Ø
A095    °
A100
/*
```

Example 12

Dynamically Create and Sort a Table, Accumulate, and Print at EOJ

Using the same general ledger transaction file which was referenced in the previous example, change the accumulation and reporting requirements.

Instead of working with only accounts matching the table, accumulate and print all accounts that are in the file. No table entries are present in the job stream; the account number entries are added to the table on the first occurrence found.

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP12:    DYNAMICALLY CREATE AND SORT A TABLE.          *
*            ACCUMULATE AND PRINT ACCOUNTS IN TABLE.       *
*                                                          *
************************************************************
*INFDISC40000500SSYS043      /* If VSE, remove * at position 1

    TABLSPEC 0200 01 04   05 08  /* Max 200 entries, acct=1-4,amt=5-12
    HDR 1A 1 $IPLDAT$  G/L ACCOUNT SUMMARY
    HDR 2A 0   ACCT                   $AMOUNT$

    ATEND 200                     /* At end-of-file on INF, branch to
    *                             200 for dumping and printing of
    *                             accumulated table entries.
100 GET                           /* Read a G/L record.
110 IF INF5-8 IS ONTABLE          /* Search table for hit on acct-nr.
       ADD INF21-28-P TO TBH5-12-P /* Found, add $ amt to table ctr
       GOTO 100.                  /* Go read next record.

 * AT THIS POINT, ACCT-NR IS NOT ONTABLE, SO WE WILL PROCEED TO ADD
 * THIS ENTRY TO THE END OF THE TABLE, AND RE-SORT THE TABLE BY ARG.

    SET TSA INITIAL               /* Set index table pointer at start.
150 IF TSA1 IS HIVALUE            /* Test for end of table (X'FF').
       GOTO 170.                  /* Eq. branch to 170.
    SET TSA UP 12                 /* Bump index to next entry.
    GOTO 150                      /* Go to start of loop.
170 MOVE INF5-8 TO TSA1-4         /* Move acct-nr from INF to table arg1-4.
    MOVE ZEROES TO TSA5-12-P      /* Move 0 to table function, pos5-12.
    SET TSA INITIAL               /* Set index table pointer at start.

    CALL TABLSORT TSA1 C'12' C'01' C'04' C'A' C'C'
    GOTO 110                      /* Go do search & add to table.

 *     EOF PROCESSING FOLLOWS
200 SET TSA INITIAL               /* Point index at table start.
210 IF TSA1 IS HIVALUE            /* Test for end of table.
       GOTO EOJ.                  /* Eq, go force EOJ.
    MOVE TSA1-4 TO PRT4           /* Move acct-nr to print.
    MOVE TSA5-12-P TO PRT12 2C    /* Move accum $ to print 2 dec & commas
    PRINT DOUBLESPACED            /* Print table entry.
    SET TSA UP 12                 /* Increment index to next table entry
    GOTO 210                      /* Go process next table entry.
9999END
```

Include some kind of checking logic to diagnose when the allocated table space has been exhausted. You should count and compare the number of table entries being added to a value equal to the number specified in the TABLSPEC statement.

Example 13

Table Load, TABLSORT, Print Various Sequences, Multiple HDR, and Various OPTION Parameter Overrides

The statements starting with statement 050 read the input data, move the input data to an edited format in the table, extract the first and last names of the employee into separate fields, and print the table entry.

When end-of-file is reached, control goes to statement 300. The data in the table is then alternately sorted (in memory by TABLSORT) and printed by last name sequence, first name sequence, gross descending sequence, and record-number sequence. The GO TO EOJ statement instructs VISION:Report to do end-of-job processing and termination. Starting with Statement 700, VISION:Report browses through the table and prints the table entries.

The OPTION SEQCHK=NO does not require sequence numbers, except on statements which are used for transfer points and also allows statements to be indented. Most likely, this is the default. The OPTION STMTEND=80 allows a VISION:Report input statement to require the entire 80 bytes on one line. Frequently, when HDR statements are used, this OPTION is specified. To allow for sequence numbers in positions 73-80 of input statements, STMTEND=71 should be specified or generated as the default. Using this option would allow a continuation in position 72, with sequence numbers in 73-80. To find out what default options have been specified, run a VISION:Report program without any OPTION statements, except LISTOPT=YES. Specifying this will print all the default options generated.

VISION:Report Statements

```
OPTION BWZ=YES,SEQCHK=NO,STMTEND=80,WSTSIZE=2000
************************************************************
*                                                          *
* SAMP13:    PAYROLL DEMO FILE IS USED AS DATA TO LOAD     *
*            A TABLE, THEN SORT THE TABLE, AND PRINT       *
*            IN VARIOUS SEQUENCES, SUCH AS LAST NAME,      *
*            FIRST NAME, GROSS DESCENDING SEQUENCE, AND    *
*            RECORD NUMBER SEQUENCE.                       *
*                                                          *
************************************************************
*INFCARD                    /* IF VSE, REMOVE * AT POSITION 1

    HDR 1A 1TABLE LOAD SEQUENCE                              PAGE$PG$
```

```
    HDR 2A 0PLANT    DEPT     EMPNO     LAST NAME   FIRST NAME        Y-T-D  G
    HDR 2B ROSS    REC-NR
    TRACE LAST50               /* SET TRACE FOR LAST 50 STMTS IF JOB BOMBS.
    TABLSPEC 0200 01 02   03 78    /* ALLOCATE 200 ENTRIES @ 80-BYTES EACH.
    SET TSA INITIAL            /* POINT TO 1ST TABLE ENTRY.
    MOVE SPACES TO WST1-50     /* CLEAR WST TO BLANKS FOR LATER USE.
    MOVE ZEROES TO WST61-63    /* ZERO AREA FOR COUNTER USE.
    ATEND 300                  /* AT EOF GO TO TABLE PROCESSING.

050 GET                        /* READ A RECORD.
    MOVE SPACES TO TSA1-80     /* CLEAR NEXT TABLE ENTRY TO BLANKS.
    MOVE INF1-2 TO TSA3        /* MOVE PLANT TO TABLE
    MOVE INF3-5 TO TSA10-12    /*      DEPT
    MOVE INF6-9 TO TSA19       /*      EMP-NR.
    MOVE INF10-25 TO WST1      /* MOVE NAME TO WORK AREA.
    WHEN WST1-16 INCLUDES C',' /* NOW SCAN FOR COMMA.
    MOVE PTR2-13 TO TSA44      /* MOVE NAME BEHIND COMMA TO TABLE.
    MOVE SPACES TO PTR1-15     /* BLANK COMMA & NAME FOLLOWING IT.<F255D>
    MOVE WST1-12 TO TSA29.     /* MOVE LAST NAME TO TABLE.<P9>
    MOVE INF26-32 TO TSA62-68  /* MOVE Y-T-D GROSS TO TABLE.<P9>
    ADD C'1' TO WST61-63       /* INCREMENT RECORD NUMBER BY 1.<F255D>
    MOVE WST61-63 TO TSA77     /* MOVE REC-NR TABLE.
    PERFORM 800 THRU 899       /* PRINT THE TABLE ENTRY.
    SET TSA UP 80              /* INCREMENT TABLE INDEX POINTER BY 80.
    GOTO 050                   /* GO READ NEXT RECORD.

300 MOVE C'END OF INF/TABLE LOAD' TO PRT1   /*
    PRINT DOUBLESPACED         /* PRINT TABLE-LOADED MESSAGE.
    SET TSA INITIAL       /* INITIALIZED TABLE INDEX POINTER TO START.
315 CALL TABLSORT TSA1 C'80' C'29' C'12' C'A' C'C'
                           /* SORT TABLE BY LAST NAME.
    MOVE SPACES                 TO HDA1 /*
    MOVE C'LAST NAME SEQUENCE    ' TO HDA1 /*
    PERFORM 700 THRU 799       /* DUMP & PRINT SORTED TABLE.
    SET TSA INITIAL            /* INITIALIZE TABLE INDEX POINT TO START.
330 CALL TABLSORT TSA1 C'80' C'44' C'12' C'A' C'C'
                           /* SORT TABLE BY FIRST NAME.
    MOVE SPACES                 TO HDA1 /*
    MOVE C'FIRST NAME SEQUENCE   ' TO HDA1             /*
    PERFORM 700 THRU 799       /* DUMP & PRINT SORTED TABLE.
    SET TSA INITIAL            /* INITIALIZE TABLE INDEX POINTER TO START.
350 CALL TABLSORT TSA1 C'80' C'62' C'07' C'D' C'C'
                           /* SORT GROSS / DESCENDING.
    MOVE SPACES                 TO HDA1 /*
    MOVE C'GROSS DESC. SEQUENCE  ' TO HDA1           /*
    PERFORM 700 THRU 799       /* DUMP & PRINT SORTED TABLE.
    SET TSA INITIAL            /* INITIALIZE TABLE INDEX POINTER TO START.
370 CALL TABLSORT TSA1 C'80' C'77' C'03' C'A' C'C'  /* SORT TABLE BY REC-NR.
    MOVE SPACES                 TO HDA1 /*
    MOVE C'REC-NR SEQUENCE      ' TO HDA1      /*
    PERFORM 700 THRU 799              /* DUMP AND PRINT SORTED TABLE.
    MOVE C'END OF TEST'        TO PRT1 /* E-X-A-M-P-L-E OF QUOTES
    PRINT DOUBLESPACED         /* PRINT THE CARTOON ENDING.
    GO TO EOJ                  /* FORCE EOJ.

700 DOHEADERS                  /* PAGE EJECT AND PRINT HEADERS.
    SET TSA INITIAL            /*
710 IF TSA1-10 IS HIVALUE      /* TEST FOR TABLE END.
        GO TO 790.             /* EQ. GO PRINT MESSAGE AND EXIT.
    PERFORM 800 THRU 899       /* PRINT CURR. TABLE ENTRY AT STAT. 800-899.
    SET TSA UP 80              /* INCREMENT TABLE INDEX POINTER TO NEXT ENTRY.
    GOTO 710                   /* GO PROCESS THIS ENTRY.
790 MOVE C'* * *  END OF TABLE * * *' TO PRT1        /*
    PRINT DOUBLESPACED        /* MOVE & PRINT ENDING MESSAGE.
799 EXIT                      /* EXIT ....
```

```
800 MOVE TSA1-80 TO PRT1     /* MOVE TABLE ENTRY TO PRINT LINE.
    PRINT                    /* PRINT IT
899 EXIT                     /* EXIT ....
9999END                      /* STATEMENTS END./* FOLLOWS, THEN INF DATA
/*
120050002CLEARY,TOM      2371160
120050008RUNNINGTREE,TOM 3252304
120050014LEANINGHORSE,C.E3096720
120050017LUBINPINSKI,CLY 3841728
120050026RODKOWSKI,GENE  3837240
120050103LONNYSTAR,RACHEL3518592
120050105ZONK,HIERONYMOUS4562800
120050129ATWATER,SCOTT   2760120
120050340CLEGHORN,DELLA  3466232
120080356CLEMENS,GARY    4206752
120080387CLEVELAND,GROVER4049672
120080399COCER,ONIES     2697288
120080410EVERS,HANK      3593392
120100423FAIR,MAXINE     4921840
120150620LAFARY,ALFRED   4453592
120150621LANDERS,CAROL   4745312
120150763LANDERS,MICHAEL 4139432
120150867LAROCHELLE,RISA 5179152
121201034LAWSON,MOLER    6066280
/*
```

```
TABLE LOAD SEQUENCE                                  PAGE   1
PLANT    DEPT     EMPNO     LAST NAME   FIRST NAME        Y-T-D  GROSS    REC-NR

  12     005      0002      CLEARY          TOM              2371160        001
  12     005      0008      RUNNINGTREE     TOM              3252304        002
  12     005      0014      LEANINGHORSE    C.E              3096720        003
  12     005      0017      LUBINPINSKI     CLY              3841728        004
  12     005      0026      RODKOWSKI       GENE             3837240        005
  12     005      0103      LONNYSTAR       RACHEL           3518592        006
  12     005      0105      ZONK            HIERONYMOUS      4562800        007

LAST NAME SEQUENCE                                   PAGE   2
PLANT    DEPT     EMPNO     LAST NAME   FIRST NAME        Y-T-D  GROSS    REC-NR

  12     005      0129      ATWATER         SCOTT            2760120        008
  12     005      0002      CLEARY          TOM              2371160        001
  12     005      0340      CLEGHORN        DELLA            3466232        009
  12     008      0356      CLEMENS         GARY             4206752        010
  12     008      0387      CLEVELAND       GROVER           4049672        011
  12     008      0399      COCER           ONIES            2697288        012
  12     008      0410      EVERS           HANK             3593392        013
  12     010      0423      FAIR            MAXINE           4921840        014

FIRST NAME SEQUENCE                                  PAGE   3
PLANT    DEPT     EMPNO     LAST NAME   FIRST NAME        Y-T-D  GROSS    REC-NR

  12     015      0620      LAFARY          ALFRED           4453592        015
  12     005      0014      LEANINGHORSE    C.E              3096720        003
  12     015      0621      LANDERS         CAROL            4745312        016
  12     005      0017      LUBINPINSKI     CLY              3841728        004
  12     005      0340      CLEGHORN        DELLA            3466232        009
  12     008      0356      CLEMENS         GARY             4206752        010
  12     005      0026      RODKOWSKI       GENE             3837240        005
  12     008      0387      CLEVELAND       GROVER           4049672        011

GROSS DESC. SEQUENCE                                 PAGE   4
PLANT    DEPT     EMPNO     LAST NAME   FIRST NAME        Y-T-D  GROSS    REC-NR

  12     120      1034      LAWSON          MOLER            6066280        019
  12     015      0867      LAROCHELLE      RISA             5179152        018
  12     010      0423      FAIR            MAXINE           4921840        014
  12     015      0621      LANDERS         CAROL            4745312        016
  12     005      0105      ZONK            HIERONYMOUS      4562800        007
  12     015      0620      LAFARY          ALFRED           4453592        015
  12     008      0356      CLEMENS         GARY             4206752        010
  12     015      0763      LANDERS         MICHAEL          4139432        017

REC-NR SEQUENCE                                      PAGE   5
PLANT    DEPT     EMPNO     LAST NAME   FIRST NAME        Y-T-D  GROSS    REC-NR

  12     005      0002      CLEARY          TOM              2371160        001
  12     005      0008      RUNNINGTREE     TOM              3252304        002
  12     005      0014      LEANINGHORSE    C.E              3096720        003
  12     005      0017      LUBINPINSKI     CLY              3841728        004
  12     005      0026      RODKOWSKI       GENE             3837240        005
  12     005      0103      LONNYSTAR       RACHEL           3518592        006
  12     005      0105      ZONK            HIERONYMOUS      4562800        007
  12     005      0129      ATWATER         SCOTT            2760120        008
```

The report output was truncated for conciseness.

Example 14

Native VSAM Using GET, QUIKIPDS/QUIKINCL, REPORT, SORT AREA with RELEASE/RETURN, DISPLAY, CALL to QUIKDATE

This example reads the AR file and builds an area for sorting (using the RELEASE verb to pass a record to the SORT and the RETURN verb to receive a sorted record and the location to which it is to be delivered). The area could have been made smaller, in order for the SORT to be more efficient, and not have to sort as big a record as the original input file. The REPORT statement is used to print a report. At EOJ time, a message displays on the console. A CALL to QUIKDATE is also demonstrated.

The OPTION UEXIT1=subrtn-exit allows you to invoke a user exit. QUIKIPDS (for MVS) or QUIKINCL (for VSE) is used in this example as a method to get file definitions for the AR file, allowing the standardization of file definitions within VISION:Report.

Note: The OPTION UEXIT1=QUIKIPDS, in conjunction with the ++INCLUDE ARDEFINE, allows you to retrieve the member from a partitioned data set (PDS) under MVS; for VSE, the OPTION UEXIT1=QUIKINCL, in conjunction with the ++INCLUDE Q.ARDEFINE accomplishes the same results with a source library. The statements contained in the ++INCLUDE member are incorporated into the VISION:Report program at that point.

Other examples that follow in this chapter, as well as some examples in the SAMPLIB, also utilize this optional feature, as it helps in standardizing data names. Additional JCL is required if QUIKIPDS/QUIKINCL is used. For MVS, a DD statement with the file name of QUIKIPDS is required to point to the data set containing the member ARDEFINE. For VSE, a DLBL statement containing the source book must be included in the LIBDEF concatenation. For further details and examples of JCL required, see the sections QUIKIPDS —PDS and PDS/E Include Subroutine (MVS Only) or QUIKINCL — Source Statement Library Routine (VSE Only) in Chapter 6.

VISION:Report Statements

```
OPTION UEXIT1=QUIKIPDS   /* If VSE, use "OPTION UEXIT1=QUIKINCL"
************************************************************
*                                                          *
* SAMP14:    NATIVE VSAM USING 'GET' FOR AR FILE.          *
*                                                          *
*            SORT AREA BY AR-ZIP, WHICH USES RELEASE/RETURN*
*                                                          *
*            CONVERT 2-DIGIT MONTH TO ALPHABETIC BY        *
*            CALLING QUIKDATE.                             *
*                                                          *
*            USER EXIT1 (UEXIT1) POINTS TO USAGE OF SUBRTN *
*            QUIKIPDS (MVS) OR QUIKINCL (VSE) TO OBTAIN    *
*            FILE DEFINITIONS FOR THE AR FILE.             *
*                                                          *
*            QUIKIPDS READS A MEMBER, INDICATED ON THE     *
*            "++INCLUDE MEMNAME" STATEMENT, FROM A PDS.    *
*            THE PDS CAN BE JUST ONE DD STATEMENT OR IT    *
*            COULD BE CONCATENATED.                        *
*                                                          *
*            QUIKINCL READS A MEMBER, INDICATED ON THE     *
*            "++INCLUDE MEMNAME" STATEMENT, FROM A VSE     *
*            SOURCE STATEMENT LIBRARY.                     *
*                                                          *
*            THE "++INCLUDE ARDEFINE" STATEMENT, WHICH     *
*            MUST START IN POSITION 1, INDICATES WHERE THE *
*            MEMBER IS TO BE INCLUDED WITHIN THE PROGRAM.  *
*                                                          *
*            ADDITIONAL DD STATEMENTS OR DLBL STATEMENTS   *
*            ARE REQUIRED IN THE JOBSTREAM IF              *
*            QUIKIPDS/QUIKINCL IS USED.                    *
*                                                          *
*            DISPLAY, REPORT VERBS ALSO USED.              *
*                                                          *
************************************************************
*INFKSDS    0352                        /* If VSE, remove * at position 1
EQU  AR-ENT-REC         INF000-000
++INCLUDE ARDEFINE                     /* If VSE, “++INCLUDE Q.ARDEFINE”
EQU  MY-CTR             WST1-3-P  ZEROS   /* Print only so much
EQU  MY-MSG             WST4-19  C'END OF SAMPLE 14'
EQU  MY-MONTH           WST20-28          /* Month spelled out
EQU  MY-YYYY            WST29-32
EQU  MY-YY1             WST29-30  C'20'
EQU  MY-YY2             WST31-32

     REPORT  AR-ZIP           (Z,  I,   P)        SPACE21
             AR-ACCOUNT       (ACCOUNT-NR)        SPACE1
             AR-ACCT-CODE                         SPACE2
             AR-PHONE         (PHONE NUMBER)      SPACE2
             AR-SSN           (SOCIAL SECURITY)   SPACE2
             MY-MONTH         (TRAN MONTH)        SPACE1
             AR-TRAN-DD       (TRAN DAY)          SPACE1
             MY-YYYY          (TRAN YEAR)

     SORT AREA     RL352  ON  AR-ZIP

010  GET INF ATEND 200
     IF AR-ZIP  IS NOT  GT  ZEROS        /* Edit out garbage
          GOTO 010.
     RELEASE INF1  TO SORT
     GOTO 010

200  RETURN  SORTED INTO  INF1
     IF   VAL196  IS EQ  TO C'E'         /* See if INF at EOF
```

```
       IF VAL200  IS EQ  TO C'E'          /* As well as sort
          GOTO 900.
     IF VAL200  IS EQ  TO C'E'            /* See if sort at EOF
         GOTO 900.
     MOVE SPACES TO MY-MONTH
     MOVE SPACES TO MY-YYYY
     IF AR-TRANS-DATE IS NOT NUMERIC
         MOVE SPACES TO AR-TRANS-DATE     /* Clear it out
         GOTO 300.                        /* Skip if date not numeric
     IF AR-TRANS-DATE IS NOT GT ZEROS
         MOVE SPACES TO AR-TRANS-DATE     /* Clear it out
         GOTO 300.                        /* Skip if date not numeric
* EXAMPLE OF CALLING "QUIKDATE" TO CONVERT TWO-DIGIT MONTH
* TO ALPHABETIC MONTH
     CALL QUIKDATE  C'09' AR-TRANS-DATE C'MMDDYY '  MY-MONTH
     IF @VAL-RETURN-CD  NOT =  ZEROS
         MOVE SPACES TO MY-MONTH          /* Clear it out
         GOTO 300.
     MOVE AR-TRAN-YY    TO  MY-YY2
     MOVE C'20'  TO MY-YY1                /* Fix prefix
300  PRINT REPORT
     ADD C'1'  TO  MY-CTR
     IF MY-CTR IS LT  P'65'
         GOTO 200.

900  DISPLAY MY-MSG
     GOTO EOJ
9999 END
```

The following JCL examples show the addition of source statements or libraries that contain the ARDEFINE member, as well as sample JCL for SORT.

VSE JCL Example

```
// JOB SAMP14
// DLBL SORTOUT,'QJ.SORTOUT',0
// EXTENT SYS001,SYSWK1,1,0,1,30
// ASSGN   SYS001,DISK,VOL=SYSWK1,SHR
// DLBL QAREWK1,'WORK',0
// EXTENT  SYS003,SYSWK1,1,0,1,60
// ASSGN   SYS003,DISK,VOL=SYSWK1,SHR
// DLBL filename,'your.VISION.lib'
// EXTENT   ,volser
// LIBDEF  PHASE,SEARCH=(lib.sublib)
// DLBL    INF,'ARFILE.VSAM',,VSAM
// EXEC QUKBJOB,SIZE=512K
 … VISION:Report statements as shown above
/*
/&
```

MVS JCL Example

```
//SAMPLE15 JOB  (800-0000,0000)
//RUNIT  EXEC QJTEST
//QJ.QUIKIPDS  DD  DISP=SHR,DSN=your.VISION.source   ARDEFINE
//QJ.SYSOUT    DD  SYSOUT=*
//QJ.SORTWK01  DD  UNIT=SYSDA,SPACE=(TRK,(5,5))
//QJ.SORTWK02  DD  UNIT=SYSDA,SPACE=(TRK,(5,5))
//QJ.SYSUT1    DD  DISP=SHR,                INF
//                DSN=ARFILE.VSAM
//QJ.SYSIN    DD  *
 ...  VISION:Report statements as shown above
/*
//
```

Example 15

Additional Working Storage and QUIKVSAM, Using OPTION, POINT, GET-UPD, ERASE, and MOVE with Quotes

When there is a shortage of working storage space, there are several solutions to choose from, such as using SAV areas, increasing the size of WST, and using a DUMMY or IGN data set/file.

In this example, assume WST has a default of 1000 bytes. By using OPTION WSTSIZE=1500, the WST is expanded from 1000 bytes to 1500 bytes. The output area has the DUMMY or IGN on the appropriate DD or file name JCL statement. This example uses the technique of instructing VISION:Report to allocate buffer space for the output OFA file, but in the JCL, the file is to be assigned IGN in VSE or given DUMMY status in an MVS environment.

This example uses QUIKVSAM to access the Accounts Receivable (AR) VSAM file, using the DD or file name of ARFILE. You must include a JCL statement with the DD or file name of ARFILE. Input records (INF) containing the account code are read, and used as a pointer to the ARFILE; records are read from the ARFILE and all records matching the account code are then deleted.

This example demonstrates various commands such as:

- OPTION (allowing for VSAM updates)
- POINT to get the VSAM file at the correct key position
- GET-UPD to allow VSAM to update the ARFILE
- ERASE to delete the VSAM record
- CLOSE to signal QUIKVSAM to close the ARFILE.

Note: The DD or file name parameter on the CALL to QUIKVSAM must be 8 bytes long, padded with spaces to the right, if necessary.

For further details on QUIKVSAM, see the section QUIKVSAM in Chapter 7.

This example also includes techniques for moving literals that include single quotes into another field. Near the end of the VISION:Report program there are a few examples that use a slightly different approach.

VISION:Report Statements

```
OPTION WSTSIZE=1500     /* DEMONSTRATE EXPANDING WORKING STORAGE SIZE
OPTION UEXIT1=QUIKIPDS   /* UEXIT1=QUIKIPDS IF MVS; VSE IS QUIKINCL

*************************************************************
*                                                           *
* SAMP15:    INCREASE WSTSIZE FROM "DEFAULT" OF 1000 TO 1500 *
*            DEMONSTRATE USAGE OF INCREASING WORKING         *
*            STORAGE BY USING "OFA" AREAS, AND ASSIGNING     *
*            OFA FILE TO 'IGNORE' IF VSE, AND 'DUMMY' IF     *
*            MVS.                                            *
*            USAGE OF QUIKVSAM, WITH OPTION, POINT,          *
*            GET-UPD, ERASE, AND CLOSE.                      *
*            SEVERAL GOOD EXAMPLES OF USING THE 'MOVE'       *
*            VERB THAT HAS QUOTES IN THE LITERAL             *
*                                                           *
*************************************************************
*INFCARD                          /* IF VSE, REMOVE * AT POSITION 1
*OFATAPE20352035NSYS010           /* IF VSE, REMOVE * AT POSITION 1

EQU  AR-ENT-REC         OFA000-000
++INCLUDE ARDEFINE     /* ++INCLUDE Q.ARDEFINE   IF VSE

EQU  QV-FEEDBACK        WST1401-1413      /* QUIKVSAM FEEDBACK AREA
EQU  QV-TYPE            WST1401           /* TYPE FILE, 'K,E,R'
EQU  QV-ACCESS          WST1402           /* TYPE ACCESS
EQU  QV-FILLER1         WST1403           /* RESERVED
EQU  QV-LRECL           WST1404-1407-B    /* RECORD LENGTH
EQU  QV-VSAMRCEC        WST1408-1409-B    /* RC & EC
EQU  QV-VSAMRC          WST1408-B         /* RETURN CODE (RC)
EQU  QV-VSAMEC          WST1409-B         /* ERROR CODE
EQU  QV-RBA             WST1410-1413-B    /* RBA

EQU  DELETE-CTR         WST1420-1422 ZEROS /* NUMBER RECORDS DELETED

     REPORT  AR-LAST-NAME
             AR-ACCOUNT       (ACCOUNT-NR)
             AR-ACCT-CODE
             AR-CITY

     CALL  QUIKVSAM  C'ARFILE  ' C'OPTION'  QV-FEEDBACK /* FEEDBACK

010  GET INF  ATEND  900     /* GET INPUT PARAMETERS FOR DELETING

* READ INPUT THAT CONTAINS A 2-BYTE FIELD OF ACCT-CODE
* DELETE ALL RECORDS FROM VSAM ARFILE THAT CONTAINS KEYS (PARTIAL)
* STARTING WITH THOSE 2 BYTES.

     CALL  QUIKVSAM  C'ARFILE  ' C'POINT'   INF1   C'KGE02'
     IF QV-VSAMRCEC  = X'0810'    /* NO RECORD FOUND
         GOTO 700.                /* TELL USER THIS
020
     CALL  QUIKVSAM  C'ARFILE  ' C'GET-UPD' AR-RECORD   /* READ IT
     IF QV-VSAMRC  NOT = X'00'            /* WE'LL ASSUME EOF
         GOTO 900.
```

```
* WE'RE ONLY GOING TO DELETE ALL RECORDS WHOSE KEY STARTS
* WITH WHAT CAME IN FROM INPUT FILE

     IF AR-KEY-ACCT-CD  =  INF1-2         /* DO WE HAVE A MATCH?
         GOTO 100.                        /* ... YES, BINGO!
     GOTO 010                             /* NOT ANY MORE
100
     PRINT REPORT
     CALL  QUIKVSAM  C'ARFILE  ' C'ERASE'              /* DELETE IT
     IF QV-VSAMRC  NOT = X'00'            /* SOME KIND OF ERROR !!
         GOTO 800.                        /* ABORT AND NOTIFY
     ADD C'1'        TO DELETE-CTR
     GOTO 020

700  MOVE C'POINT FAILED. KEY='   TO PRT1
     MOVE INF1-2                  TO PRT20
     PRINT
     PRINTHEX QV-FEEDBACK
     GOTO 010

800  MOVE C'ERROR DURING ERASE'   TO PRT1
     MOVE AR-KEY                  TO PRT20
     PRINT
     PRINTHEX QV-FEEDBACK
     GOTO 010

900  CALL  QUIKVSAM  C'CLOSE'             /* WRAP-UP TIME

* EXAMPLES OF "MOVE" WITH QUOTES INTO OUTPUT FIELD
* STATEMENT 901-903 CREATES THE SAME RESULTS AS STATEMENTS 921-925
* STATEMENTS 921-925 IS THERE JUST FOR DEMONSTRATION PURPOSES
* AND A REPEAT OF STATEMENTS 901-903

901  MOVE C'NUMBER OF ''DELETED'' RECORDS:'    TO PRT1
902  MOVE DELETE-CTR                   TO PRT30 0C
903  PRINT

921  MOVE C'NUMBER OF '                TO PRT1
922  MOVE C'''DELETED'''               TO PRT11
923  MOVE C' RECORDS:'                 TO PRT20
924  MOVE DELETE-CTR                   TO PRT30 0C
925  PRINT

     MOVE C'IF MA & TO KEYS ARE USED,' TO PRT1
     MOVE C' THERE ARE 68 RECORDS DELETED' TO PRT26
     PRINT DOUBLESPACED
     GOTO EOJ

9999END
```

VSE JCL Example

```
// JOB QJVSAM
// DLBL filename,'your.VISION.lib'
// EXTENT   ,volser
// LIBDEF  PHASE,SEARCH=(lib.sublib)
// LIBDEF SOURCE,SEARCH=(lib.sublib)          Contains Q.ARDEFINE
// DLBL  ARFILE,'ARFILE.VSAM',,VSAM           Using QUIKVSAM
// ASSGN  SYS010,IGN                          Assign OFA to ignore
// EXEC  QUKBJOB
  ...  VISION:Report statements as shown above
MA
TO
/*
/&
```

MVS JCL Example

```
//QJVSAM   JOB  (800-0000,0000),'USE QUIKVSAM'
//STEP1   EXEC  PGM=QUIKJOB,REGION=512K
//STEPLIB  DD  DISP=SHR,DSN=your.VISION.loadlib
//SYSPRINT DD  SYSOUT=*
//QUIKIPDS DD  DISP=SHR,DSN=your.VISION.source Contains ARDEFINE
//ARFILE   DD  DISP=SHR,DSN=ARFILE.VSAM        Using QUIKVSAM
//SYSUT2   DD  DUMMY,                          OFA
//             DCB=(BLKSIZE=2035,LRECL=2035,RECFM=F), etc
//SYSIN    DD  *
  ...  VISION:Report statements as shown above
//SYSUT1   DD  *
MA
TO
/*
//
```

Example 16

TABLSPEC, Indexing, Table ACCUM, BREAK, Summary Output to Disk

A company wants an analysis to be done on selected accounts and departments in each of its plants. The information is contained in a cost distribution summary file that was designed in such an efficient manner as to make it difficult to use with conventional report listing programs. The record format is variable, with the first 40 positions of fixed data and the remainder being a variable number of trailer segments consisting of 20 bytes each. The maximum record length is 1040 bytes.

The task is to index through the trailer segments selecting account data which matches a table entry. On a matching entry accumulate the dollar amount in the table counter. On a break in plant or department fields write to an output file the table entries that contain non-zero amounts along with plant and department.

VISION:Report Statements

```
OPTION SEQCHK=NO

EQU PLANT-IN  INF1-2
EQU DEPT-IN   INF3-5

EQU ACCT-IN   PTA1-2
EQU AMT-IN    PTA3-9

EQU PLANT-OUT OFA1-2
EQU DEPT-OUT  OFA3-5
EQU ACCT-OUT  OFA6-7
EQU AMT-OUT   OFA8-18

    TABLSPEC 0050 01 02   03 06 LIST /* TABLE ARG=1-2, FUNCTION 3-8.

    BREAK 1 DEPT-IN       /* SPECIFY FIELDS TO DO BREAK COMPARISONS
    BREAK 2 PLANT-IN      /* NOTE, NO 'ACCUM' BEING USED
*       LOOP THROUGH TABLE & ZAP PACKED ZERO OUT COUNTER USE
    SET TSA INITIAL              /* SET INDEX TABLE POINTER TO START.

10  IF TSA1 IS HIVALUE           /* TEST FOR END OF TABLE.
       GOTO 20.                  /* EQ, GO TO STAT. 20 TO START.
    MOVE ZEROS TO TSA3-8-P       /* PLACE ZEROS IN TABLE ENTRY,
                                 /* POS 3-8, PKD.
    SET TSA UP 8                 /* INCR. INDEX TO NEXT TABLE ENTRY.
    GOTO 10                      /* GO TO START OF LOOP.

20  GET                          /* READ A RECORD.
    CHECKBREAKS ON BREAKS PERFORM 40 THRU 50 /* ACTUAL BREAK COMPARE
*       AND ON A BREAK, CONTROL WILL GO TO STAT. 40-50 FOR PROCESSING
    MOVE PLANT-IN TO WST1        /* SAVE PLT # FOR OUTPUT RECORDS.
    MOVE DEPT-IN  TO WST3        /* SAVE DEPT # FOR OUTPUT RECORDS.
    *
    SET PTA INF6
30  IF ACCT-IN IS BLANK          /* TEST FOR END OF SEGMENTS
    GOTO 20.                    /* EQ, GO READ NEXT RECORD.
```

```
    IF ACCT-IN IS ONTABLE    /* DO TABLE LOOKUP FOR ACCT IN THIS SEGMENT
    ADD AMT-IN TO TBH3-8-P.  /* FOUND, ADD AMT INTO TABLE ENTRY COUNTR
    SET PTA UP 9              /* INCREMENT INDEX POINTER TO NEXT SEGMNT
    GOTO 30                   /* GO PROCESS NEXT SEGMENT.

*  THE FOLLOWING WILL OUTPUT RECORDS FROM TABLE DATA ON A CONTROL BRK
40  SET TSA INITIAL          /* SET INDEX TABLE POINTER TO START.
45  IF TSA1 IS HIVALUE         /* TEST FOR END OF TABLE.
       GOTO 50.                /* EQ, ALL ENTRIES DONE
    IF TSA3-8-P IS ZERO        /* CHK IF THIS ENTRY IS ZERO.
       GOTO 48.                /* EQ, BYPASS ON ZERO CONDITION.
    MOVE WST1-2 TO PLANT-OUT   /* MOVE PLANT TO OUTPUT.
    MOVE WST3-5 TO DEPT-OUT    /*      DEPT
    MOVE TSA1-2 TO ACCT-OUT    /* ACCT-NR
    MOVE TSA3-8-P TO AMT-OUT   /* AMT
    WRITE OFA                  /* WRITE THE RECORD.
    MOVE ZEROES TO TSA3-8-P    /* ZAP ZEROS TO TABLE ENTRY FOR NEXT USE.
48  SET TSA UP 8               /* INCR. TABLE INDEX TO NEXT ENTRY.
    GOTO 45                    /* GO PROCESS NEXT TABLE ENTRY.
50  EXIT                       /* GO BACK TO NORMAL PROCESSING.
800 GOTO EOJ
9999END                        /* STATEMENTS END, TABLE FOLLOWS.
AA
LL
MM
TT
VV
(END OF SOURCE)
(INPUT DATA)
08005AA0006682XX0000013RR0037810DD0936100GG0602100YY0702700NN0080880
08005MM0000331SS0004511
08007VV0209085SS0004902CC0004915EE0000050HH0000002AA0092003ZZ0008100
08008RR0017171MM0000370LL0008899CC0001572YY0000665HH0000790
08012AA0449121BB0000412
08033PP0009099BB0003954DD0055557LL0038271RR0018904
12005DD0040933ZZ00000001WW0007119AA0059924EE0018663
13005ZZ0202020WW0198333QQ0005140DD0000338
13006SS0002132HH0004784MM0000998EE0001670GG0096461RR0011188II0003180
13006JJ0039700EE0082842SS0091532
18022BB0001001KK0029100II0001099UU0005989JJ0210000FF0770000

(OUTPUT DATA)
08005AA00000006682
08005MM00000000331
08007AA00000092003
08007VV00000209085
08008LL00000008899
08008MM00000000370
08012AA00000449121
08033LL00000038271
12005AA00000059924
13006MM00000000998
```

Example 17

ACCUM Counts, Amounts Using CTR-NO, BREAK, CHECKBREAKS, QUIKIPDS, LIMITREADS, PAGETOTALS, REPORT, SORT, and Numerous IF Statements While Validating

This example demonstrates how to read the ARFILE, breaking on account code within state. This example also shows the usage of PAGETOTALS and LIMITREADS, along with numerous IF statements to validate numeric fields and skip spaces in customer name or state. The file is sorted first and a report is printed using the REPORT statement. A break with a final total is also demonstrated.

VISION:Report Statements

```
OPTION UEXIT1=QUIKIPDS   /* UEXIT1=QUIKIPDS IF MVS; VSE IS QUIKINCL
************************************************************
*                                                          *
* SAMP17:    ACCUM COUNTS, AMOUNTS USING CTR-NO, BREAKS    *
*            (INCLUDING BREAK WITH 'F' (FINAL),            *
*            CHECKBREAKS, QUIKIPDS/QUIKINCL, LIMITREADS,   *
*            PAGETOTALS, REPORT, SORT, AND                 *
*            NUMEROUS 'IF' STATEMENTS DURING VALIDATION.   *
*                                                          *
*   NOTE: YOU MAY GET AN ERROR .                          *
*         CA-SORT HAS AN OPTION CALLED 'CHECK=' WHICH      *
*         MAY BE SET TO 'Y' OR 'N'. IF THE                 *
*         INSTALLATION DEFAULT IS 'Y' THEN WHENEVER        *
*         ALL THE INSERTED RECORDS ARE NOT DELETED THE     *
*         SORT TASK WILL END WITH RETURN CODE X'0010'.     *
*         INSERT 'CHECK=N' INTO THE OPTION STRING.         *
*                                                          *
*          =====>  THE ERROR CAN BE IGNORED.               *
************************************************************
*INFDISC52800352SSYS005            /* IF VSE, REMOVE * AT POSITION 1
EQU  AR-ENT-REC         INF
++INCLUDE ARDEFINE /* ++INCLUDE Q.ARDEFINE   IF VSE
EQU  CUST-NAME          WST1-15           /* SHORTEN CUSTOMER NAME
EQU  IPAY               WST16-21   2C     /*
EQU  IBAL               WST22-27   2C     /*
EQU  BAL                WST28-33   2C     /*

     TITLE 'SAMPLE17-PAGETOTALS, ACCUM W/CTR, BREAK, CHECKBREAKS'

     REPORT   AR-STATE CUST-NAME IPAY IBAL BAL AR-ACCT-CODE

     PAGETOTALS

     LIMITREADS 100 INF      /* DON'T DO ANY MORE THAN THIS
     SORT FILE INF ON AR-STATE AR-ACCT-CODE

  BREAK 1  AR-ACCT-CODE SB 1 SA 2 PRINT C'ACCT CODE TOT'
  BREAK 2  AR-STATE     SB 1 SA 2 PRINT C'STATE TOTALS'
  BREAK F  AR-ACCT-CODE SB 2 SA 2 PRINT C'FINAL TOTALS' IN TOT POS 5

010  GET INF  ATEND EOJ
```

```
    IF AR-CUST-NAME   EQ  SPACES         /* KICK OUT BLANK NAMES
        GOTO 010.
    IF AR-STATE       EQ  SPACES    OR   /* KICK OUT BLANK STATES
       IF AR-ZIP      NOT  NUMERIC       /* OR NON-NUMERIC ZIP-CODES
           GOTO 010.
    IF AR-INSTL-PAY   NOT  NUMERIC       /* ENSURE GOOD FIELD
        GOTO 010.
    IF AR-INSTL-BAL   NOT  NUMERIC       /* ENSURE GOOD FIELD
        GOTO 010.
    IF AR-BALANCE     NOT  NUMERIC       /* ENSURE GOOD FIELD
        GOTO 010.
    MOVE AR-CUST-NAME   TO  CUST-NAME    /* MAKE IT SHORTER
    MOVE AR-INSTL-PAY   TO  IPAY         /*  OR GIVE
    MOVE AR-INSTL-BAL   TO  IBAL         /*  A DIFFERENT
    MOVE AR-BALANCE     TO  BAL          /* NAME

    CHECKBREAKS

  ACCUM AR-INSTL-PAY  IN A 5 BYTE CTR, ON BREAKS PRINT IN POS 26 2C
  ACCUM AR-INSTL-BAL  IN A 5 BYTE CTR, ON BREAKS PRINT IN POS 40 2C
  ACCUM AR-BALANCE    IN A 5 BYTE CTR, ON BREAKS PRINT IN POS 54 2C

    PRINT REPORT

    GOTO 010

9999END
```

Example 18

ACCUM Using CTR, BREAK, and CHECKBREAKS

This example produces a report by department within plant. It reflects department totals, plant totals, and grand totals.

The requirements are:

- Calculate and total this week's gross.
- Calculate and total this week's FICA.
- List various fields.
- Total previous YTD gross.
- Report only departments 005 and 010.

The input record is 80 positions long and resides on disk. The format of the file is:

```
COL         CONTENT
1-2         Plant Number
3-5         Department Number
6-9         Employee Number
10-25       Name
26-31       Date of Birth (YYMMDD)
32-37       Date of Employment (YYMMDD)
38-39       Education
40-45       Skill Codes
46-49       Hourly Rate xx.xx
50-53       Hours Worked xx.xx
54-58       Previous YTD Gross xxxxxx.xx Packed Decimal
59-63       YTD State Tax xxxxx.xx Fixed Point Binary
64-69       YTD FICA xxxx.xx EBCDIC
70-80       Unused
```

VISION:Report Statements

```
    OPTION LISTOPT=YES,STMTEND=80
************************************************************
*                                                          *
* SAMP18:    PRODUCE REPORT BY DEPARTMENT WITHIN PLANT.    *
*            ACCUM USING CTR, BREAKS, AND CHECKBREAKS.     *
*                                                          *
************************************************************
*INFCARD                          /* IF VSE, REMOVE * AT POSITION 1

EQU GROSS-PAY         WST1-4-P
EQU CURR-FICA         WST9-11-P
EQU WST-PLANT-DEPT    WST101-105
EQU PLANT-DEPT        INF1-5
EQU PLANT             INF1-2
EQU DEPARTMENT        INF3-5
EQU EMP-NR            INF6-9
EQU EMP-NAME          INF10-25
EQU BIRTH-DATE        INF26-31      /* YYMMDD
```

```
EQU EMPLY-DATE        INF32-37      /* YYMMDD
EQU EDUCATION         INF38-39
EQU HOURLY-RATE       INF46-49      2
EQU HOURS-WORKED      INF50-53      2
EQU PREV-YTD-GROSS    INF54-58-P    2C
EQU YTD-STATE-TAX     INF59-63-B    2C
EQU YTD-FICA          INF64-69      2C
000 HDR 1A 1  $IPLDAT$                          PAYROLL DEMO WITH SUMMARY
001 HDR 1B                    PAGE $PG$
002 HDR 2A 0                                     DATE OF     DATE OF
003 HDR 2B PREVIOUS     HRLY    THIS        YTD
004 HDR 3A  PLT DEPT EMP EMPLOYEE.NAME           .BIRTH..  EMPLOYMENT EDUC YT
005 HDR 3B D GROSS WORK RATE WEEK GROSS STATE TAX YTD FICA CURR FICA
010 BREAK 1 DEPARTMENT SB 1 SA 1 PRINT C'DEPT TOTS' IN TOT POS 011
020 BREAK 2 PLANT SB 1 SA E PRINT C'PLANT TOTS' IN TOT POS 011
050 GET
    IF DEPARTMENT IS EQ C'005' OR
       IF DEPARTMENT IS EQ C'010'
       GO TO 060.
    GO TO 050
060 CHECKBREAKS
    MOVE PLANT-DEPT TO WST-PLANT-DEPT  /* STORE THE PLANT AND DEPT NR.
    MOVE PLANT TO PRT2                 /* MOVE PLANT NR TO PRINT.
    MOVE DEPARTMENT TO PRT6            /* MOVE DEPT NR TO PRINT.
    MOVE EMP-NR TO PRT10    0     /* MOVE EMPLOYEE NR TO PRINT, ZERO-SUPPRESS.
    MOVE EMP-NAME TO PRT15             /* MOVE EMP NAME TO PRINT.
    MOVE INF26-31 TO PRT32  D          /* MOVE DATE OF BIRTH, EDITED.
    MOVE INF32-37 TO PRT42  D          /* MOVE DATE OF EMPLOYMENT, EDITED.
    MOVE EDUCATION TO PRT53            /* MOVE EDUCATION TO PRINT.
    MOVE PREV-YTD-GROSS TO PRT56       /* MOVE PREVIOUS YTD GROSS TO PRINT.
    MOVE HOURS-WORKED TO PRT69         /* MOVE HOURS WORKED TO PRINT.
    MOVE HOURLY-RATE TO PRT75  2       /* MOVE HOURLY RATE TO PRINT.
    MULT HOURS-WORKED 2D BY HOURLY-RATE 2D GIVING GROSS-PAY 2DR   /*
*****                                  /* ABOVE CALCULATES GROSS PAY.
    MOVE YTD-STATE-TAX TO PRT90 2C     /* BINARY YTD STATE TAX, CVT TO PKD DEC.
    MOVE GROSS-PAY  TO PRT83 2C        /* MOVE THIS WEEKS GROSS TO PRINT.
    MOVE YTD-FICA TO PRT110 2C         /* MOVE YTD FICA TO PRINT.
    MULT GROSS-PAY 2D BY C'058' 3D GIVING CURR-FICA 2DR /*
*****                                  /* ABOVE CALCULATES FICA.
    MOVE CURR-FICA TO PRT125 2         /* MOVE CURRENT FICA TO PRINT.
    PRINT
    ACCUM PREV-YTD-GROSS IN A 5 BYTE CTR, ON BREAKS PRINT IN POS 052 2C
    ACCUM GROSS-PAY  IN A 4 BYTE CTR, ON BREAKS PRINT IN POS 078 2C
    ACCUM CURR-FICA  IN A 4 BYTE CTR, ON BREAKS PRINT IN POS 110 2C
    GO TO 050
99999999END
120050002CLEARY,TOM      54020385081912     15854000 ƒÉ     ®d050631
120050008RUNNINGTREE,TOM 58051088051516     21744100  ê <   |ÿ069446
120050014LEANINGHORSE,C.E67082593070314     20704005  oÊ    “þ066124
120050017LUBINPINSKI,CLY 49033170110516     25684500 ˜ Êð   ]ƒ082031
120050026RODKOWSKI,GENE  51101683081017     25654000 ˜ „    [ê081936
120050103LONNYSTAR,RACHEL66112395011616     23524000 • ßŒ   Øí075132
120050105ZONK,HIERONYMOUS75042095121318     30504300 áÂØ    ž*097429
120050129ATWATER,SCOTT   72072792030912     18455100  -     6(058936
120050155MERCURY,HARRY   69080491061113     20424800  ècŒ   ‹Õ065229
120050158WINTERGARTEN,L.R55102076090416     25374200  n•Œ   ¥¥081041
120050176LADRIGANSKI,SUE 64051488051412     22354000 “äî    -:071394
120050194JONES,LYLA      73120292031116     26203800 ™ ê    I.083693
120050340CLEGHORN,DELLA  50071079051716     23174000 ”Ãƒ‹   Îò074014
120080356CLEMENS,GARY    52111573091116     28123200 â†ÍŒ   Ùj089826
120080387CLEVELAND,GROVER65090792080214     27074600  ñÂŒ   \Û086472
120080399COCER,ONIES     71123194121914     18034400  pˆð   ²ù057594
120080410EVERS,HANK      66071085081116     24024000 •l™Œ   ý1076729
120100423FAIR,MAXINE     65052285090118     32904200 ñ d    "·105095
120100452KIRBY,THOMAS F  76091095100612     17954500  e     Y^057339
```

```
120100578KUEHN,JOHN       75102095112312      26804000  ˆ     R~085609
120100594LADD,IDA         64110487123116      21834000  ÁÎð   êê069733
120150620LAFARY,ALFRED    63121788010216      29774200 àëßŒ   Š'095097
120150621LANDERS,CAROL    66012990021417      31723600 åá‘Œ   ¬l101326
120150763LANDERS,MICHAEL  54021286031512      27674200  ™äŒ   1&088388
120150867LAROCHELLE,RISA  58030686040116      34624400 é`…Œ   Þg110590
121201034LAWSON,MOLER     70041993051418      40555400 -Ãˆ   œ|9129532
```

09/17/02 PAYROLL DEMO WITH SUMMARY PAGE 1

PLT	DEPT	EMP	EMPLOYEE.NAME	DATE OF BIRTH..	DATE OF EMPLOYMENT	EDUC	YT D GROSS	WORK WEEK	HRLY RATE	THIS WEEK GROSS	YTD STATE TAX	YTD FICA	CURR FICA
12	005	2	CLEARY,TOM	54/02/03	85/08/19	12	23,711.60	40.00	15.85	634.00	1,104.68	506.31	36.77
12	005	8	RUNNINGTREE,TOM	58/05/10	88/05/15	16	32,523.04	41.00	21.74	891.34	1,515.19	694.46	51.70
12	005	14	LEANINGHORSE,C.E	67/08/25	93/07/03	14	30,967.20	40.05	20.70	829.04	1,442.70	661.24	48.08
12	005	17	LUBINPINSKI,CLY	49/03/31	70/11/05	16	38,417.28	45.00	25.68	1,155.60	1,789.79	820.31	67.02
12	005	26	RODKOWSKI,GENE	51/10/16	83/08/10	17	38,372.40	40.00	25.65	1,026.00	1,787.70	819.36	59.51
12	005	103	LONNYSTAR,RACHEL	66/11/23	95/01/16	16	35,185.92	40.00	23.52	940.80	1,639.25	751.32	54.57
12	005	105	ZONK,HIERONYMOUS	75/04/20	95/12/13	18	45,628.00	43.00	30.50	1,311.50	2,125.72	974.29	76.07
12	005	129	ATWATER,SCOTT	72/07/27	92/03/09	12	27,601.20	51.00	18.45	940.95	1,285.89	589.36	54.58
12	005	155	MERCURY,HARRY	69/08/04	91/06/11	13	30,548.32	48.00	20.42	980.16	1,423.19	652.29	56.85
12	005	158	WINTERGARTEN,L.R	55/10/20	76/09/04	16	37,953.52	42.00	25.37	1,065.54	1,768.18	810.41	61.80
12	005	176	LADRIGANSKI,SUE	64/05/14	88/05/14	12	33,435.60	40.00	22.35	894.00	1,557.70	713.94	51.85
12	005	194	JONES,LYLA	73/12/02	92/03/11	16	39,195.20	38.00	26.20	995.60	1,826.03	836.93	57.74
12	005	340	CLEGHORN,DELLA	50/07/10	79/05/17	16	34,662.32	40.00	23.17	926.80	1,614.85	740.14	53.75
			DEPT TOTS				448,201.60			12,591.33		730.29	
12	010	423	FAIR,MAXINE	65/05/22	85/09/01	18	49,218.40	42.00	32.90	1,381.80	2,292.99	1,050.95	80.14
12	010	452	KIRBY,THOMAS F	76/09/10	95/10/06	12	26,853.20	45.00	17.95	807.75	1,251.04	573.39	46.85
12	010	578	KUEHN,JOHN	75/10/20	95/11/23	12	40,092.80	40.00	26.80	1,072.00	1,867.85	856.09	62.18
12	010	594	LADD,IDA	64/11/04	87/12/31	16	32,657.68	40.00	21.83	873.20	1,521.46	697.33	50.65
			DEPT TOTS				148,822.08			4,134.75		239.82	
			PLANT TOTS				597,023.68			16,726.08		970.11	

Example 19

ACCUM Using CTA-CTC, BREAK, CHECKBREAKS, and Summary Output, PUNCH

This example is similar to Example 18, except a department record is produced on the card punch (tape or disk file can also be used).

The department summary card is to contain:

```
COL         CONTENT
1-2         Plant Number
3-5         Department Number
30-40       Gross Pay for the Department this pay period
42-50       FICA for the Department for this pay period
52-60       Hours worked by the Department this pay period
```

The break routine or total time exit allows the accumulated data to be formatted and written at department break-time.

A department number, department name table will be used to get the department name. (Normally the department table would be in a consolidated table disk for installation; if this were the case, a QUIKVSAM-generated random read routine could be used by a CALL to obtain the department name.)

The program used in the previous example will be altered slightly to use named accumulators (such as, CTA, CTB) so that the accumulated data may be referenced at the appropriate break routines.

EQU statements will be used to make the various data names and working storage names more meaningful.

The summary records will be hex printed to show the content and sequence of occurrence.

VISION:Report Statements

```
OPTION LISTOPT=NO,STMTEND=80,TRACECT=1
*INFDISC32000080SSYS005    /* IF VSE, REMOVE * AT POS 1
    *-------------------------------------------------------*
    *                                                       *
    * SAMP19:   ACCUM USING CTA-CTC, BREAK, CHECKBREAKS,    *
    *           AND SUMMARY OUTPUT, PUNCH.                  *
    *                                                       *
    *                                                       *
    *           PRODUCE REPORT BY DEPARTMENT WITHIN PLANT.  *
    *           HAVE BREAK ROUTINE AND ACCUMULATE TOTALS    *
    *           TO BE WRITTEN AT DEPARTMENT BREAK TIME.     *
    *                                                       *
    *-------------------------------------------------------*
EQU GROSS-PAY             WST1-4-P
EQU CURR-FICA             WST9-11-P
EQU WST-PLANT-DEPT        WST101-105
EQU PLANT-DEPT            INF1-5
```

```
EQU PLANT                INF1-2
EQU DEPARTMENT           INF3-5
EQU EMP-NR               INF6-9
EQU EMP-NAME             INF10-25
EQU BIRTH-DATE           INF26-31          /* YYMMDD
EQU EMPLY-DATE           INF32-37          /* YYMMDD
EQU EDUCATION            INF38-39
EQU HOURLY-RATE          INF46-49      2
EQU HOURS-WORKED         INF50-53      2
EQU PREV-YTD-GROSS       INF54-58-P    2C
EQU YTD-STATE-TAX        INF59-63-B 2C
EQU YTD-FICA             INF64-69   2C
    HDR 1A 1  $IPLDAT$    SAMPLE 19:        PAYROLL DEMO WITH SUMMARY
    HDR 1B                     PAGE $PG$
    HDR 2A 0                                DATE OF     DATE OF      PREVIOUS
    HDR 2B       HRLY      THIS      YTD
    HDR 3A  PLT DEPT EMP EMPLOYEE NAME     BIRTH..  EMPLOYMENT EDUC YTD GROSS
    HDR 3B WORK  RATE  WEEK GROSS STATE TAX  YTD FICA   CURR FICA
    TABLSPEC 0200 01 03   04 16
    BREAK 1 DEPARTMENT  SB 1 SA 1 PRINT C'DEPT TOTS'  IN TOT POS 011
    BREAK 2 PLANT       SB 1 SA E PRINT C'PLANT TOTS' IN TOT POS 011
050 GET
    IF DEPARTMENT IS EQ C'005' OR
       IF DEPARTMENT IS EQ C'010'
       GO TO 060.
    GO TO 050
060
    CHECKBREAKS ON BREAKS PERFORM 600 THRU 699
    MOVE PLANT-DEPT TO WST-PLANT-DEPT      /* STORE THE PLANT AND DEPT NR.
    MOVE PLANT TO PRT2                     /* MOVE PLANT NR TO PRINT.
    MOVE DEPARTMENT TO PRT6                /* MOVE DEPT NR TO PRINT.
    MOVE EMP-NR TO PRT10 0          /* MOVE EMPLOYEE NR TO PRINT/ZERO-SUPPRESS.
    MOVE EMP-NAME TO PRT15                 /* MOVE EMPLOYEE NAME TO PRINT.
    MOVE BIRTH-DATE  TO PRT32  D           /* MOVE DATE OF BIRTH, EDITED
    MOVE EMPLY-DATE  TO PRT42  D           /* MOVE DATE OF EMPLOY,EDITED
    MOVE EDUCATION TO PRT52                /* MOVE EDUCATION TO PRINT.
    MOVE PREV-YTD-GROSS TO PRT55 2C        /* MOVE PREVIOUS YTD GROSS TO PRINT.
    MOVE HOURS-WORKED TO PRT66 2           /* MOVE HOURS WORKED TO PRINT.
    MOVE HOURLY-RATE TO PRT72  3           /* MOVE HOURLY RATE TO PRINT.
    MULT HOURS-WORKED 2D BY HOURLY-RATE 3D GIVING GROSS-PAY 2DR
    *                                      /* ABOVE CALCULATES GROSS PAY.
    MOVE GROSS-PAY TO PRT78 2C             /* MOVE THIS WEEKS GROSS TO PRINT.
    MOVE YTD-STATE-TAX TO PRT89 2C         /* BINARY YTD STATE TAX
    MOVE YTD-FICA TO PRT102 2C             /* MOVE YTD FICA TO PRINT.
    MULT GROSS-PAY 2D BY C'058' 3D GIVING CURR-FICA 2DR
    *                                      /* ABOVE CALCULATES FICA.
    MOVE CURR-FICA TO PRT113 2             /* MOVE CURRENT FICA TO PRINT.
    PRINT
  ACCUM PREV-YTD-GROSS IN A 5 BYTE CTA, ON BREAKS PRINT IN POS 052 2C
  ACCUM GROSS-PAY      IN A 4 BYTE CTB, ON BREAKS PRINT IN POS 078 2C
  ACCUM CURR-FICA      IN A 4 BYTE CTC, ON BREAKS PRINT IN POS 110 2C
    ADD HOURS-WORKED TO CTD1-8-P        /* ADD HOURS WORKED, DON'T PRINT TOTAL.
    GO TO 050
600
    IF VAL180 IS NOT EQ C'1'          /* IF NOT MINOR/DEPT BREAK - DO NOTHING.
       GOTO 699.
    MOVE WST-PLANT-DEPT TO PUN1-5     /* MOVE STORED PLT/DEPT TO OUTPUT.
    MOVE CTB1-8-P TO PUN30-40         /* MOVE THIS DEPT'S GROSS TO SUM REC.
    MOVE CTC1-8-P TO PUN42-50         /* MOVE FICA FOR DEPT TO SUM REC.
    MOVE CTD1-8-P TO PUN52-60         /* MOVE DEPT'S HOURS WORKED TO SUM REC.
    IF PUN3-5 IS ONTABLE              /* FIND DEPARTMENT NAME IN TABLE.
       MOVE TBH4-19 TO PUN10-25.      /* MOVE DEPT NAME FROM TABLE TO RECORD.
    PUNCH
699 EXIT
9999END
```

```
  10/19/00                              PAYROLL DEMO WITH SUMMARY                                    PAGE   1
                                                 DATE OF    DATE OF          PREVIOUS          HRLY         THIS      YTD
PLT DEPT EMP  EMPLOYEE NAME        BIRTH     EMPLOYMENT ED   YTD GROSS WORK    RATE  WEEK GROSS  STATE TAX  YTD FICA  CURR FICA
12  005    2  CLEARY, TOM       54/02/03  85/08/19   12   23,711.60   40.00   15.85    63.40   1,104.68   506.31       3.68
12  005    8  RUNNINGTREE, TOM  58/05/10  88/05/15   16   32,523.04   41.00   21.74    89.13   1,515.19   694.46       5.17
12  005   14  LEANINGHORSE, C.E 67/08/25  93/07/03   14   30,967.20   40.05   20.70    82.90   1,442.70   661.24       4.81
12  005   17  LUPINPINSKI, CLY  49/03/31  70/11/05   16   38,417.28   45.00   25.68   115.56   1,789.79   820.31       6.70
12  005   26  RODKOWSKI,GENE    51/10/16  83/08/10   17   38,372.40   40.00   25.65   102.60   1,787.70   819.36       5.95
12  005  103  LONNYSTAR,RACHEL  66/11/23  95/01/16   16   35,185.92   40.00   23.52    94.08   1,639.25   751.32       5.46
12  005  105  ZONK,HIERONYMOUS  75/04/20  95/12/13   18   45,628.00   43.00   30.50   131.15   2,125.72   974.29       7.61
12  005  129  ATWATER,SCOTT     72/07/27  92/03/09   12   27,601.20   51.00   18.45    94.10   1,285.89   589.36       5.46
12  005  155  MERCURY,HARRY     69/08/04  91/06/11   13   30,548.32   48.00   20.42    98.02   1,423.19   652.29       5.69
12  005  158  WINTERGARTEN,L.R  55/10/20  76/09/04   16   37,953.52   42.00   25.37   106.55   1,768.18   810.41       6.18
12  005  176  LADRIGANSKI,SUE   64/05/14  88/05/14   12   33,435.60   40.00   22.35    89.40   1,557.70   713.94       5.19
12  005  194  JONES,LYLA        73/12/02  92/03/11   16   39,195.20   38.00   26.20    99.56   1,826.03   836.93       5.77
12  005  340  CLEGHORN,DELLA    50/07/10  79/05/17   16   34,662.32   40.00   23.17    92.68   1,614.85   740.14       5.38
                   PUN1-80 12005                              00000125913 000007305 000054805
                         FFFFF4444444444444444444444444FFFFFFFFFFF4FFFFFFFF4FFFFFFFF44444444444444444444
                         12005000000000000000000000000000000125913000000730500000548050000000000000000000
                        01..05...10...15...20...25...30...35...40...45...50...55...60...65...70...75...80
           DEPT TOTS                                  448,201.60              1,259.13                        73.05
12  010 423 FAIR,MAXINE         65/05/22  85/09/01   18   49,218.40   42.00   32.90   138.18   2,292.99 1,050.95       8.01
12  010  452 KIRBY,THOMAS F     76/09/10  95/10/06   12   26,853.20   45.00   17.95    80.78   1,251.04   573.39       4.69
12  010  578 KUEHN,JOHN         75/10/20  95/11/23   12   40,092.80   40.00   26.80   107.20   1,867.85   856.09       6.22
12  010  594 LADD,IDA           64/11/04  87/12/31   16   32,657.68   40.00   21.83    87.32   1,521.46   697.33       5.06
                  PUN1-80 12010                     00000041348 000002398 000016700
                         FFFFF4444444444444444444444444FFFFFFFFFFF4FFFFFFFF4FFFFFFFF44444444444444444444
                         12010000000000000000000000000000000413480000002398000001670000000000000000000000
                        01..05...10...15...20...25...30...35...40...45...50...55...60...65...70...75...80
           DEPT TOTS                                  148,822.08                413.48                          23.98
           PLANT TOTS                                 597,023.68              1,672.61                          97.03
```

Example 20

ACCUM, BREAK, and CHECKBREAKS

This example demonstrates some of the enhancements Options RPTSPCE and MOVCVTX, along with an increased PRTSIZE. Input file INX and output file OFF are also used. Expanded maximum lengths for numeric EBCDIC, packed, and binary fields are shown, as are sequence numbers greater than 4 digits.

VISION:Report Statements

```
OPTION PRTSIZE=161,STMTEND=80    /* PRTSIZE increased
OPTION TRACECT=1
OPTION RPTSPCE=2                 /* Default space between fields
OPTION MOVCVTX=YES               /* Convert moving x'...' to receiving field format
*INXDISC32000080SSYS003          /* IF VSE, REMOVE * AT POS 1
*OFFDISC60046004SSYS005          /* IF VSE, REMOVE * AT POS 1
    *-------------------------------------------------------*
    * SAMP20:   PRODUCE A QUARTERLY EARNINGS REPORT       *
    *           FROM THE PERSONNEL MASTER FILE (INX).     *
    *           WRITE OUT A QUARTERLY FILE (OFF).         *
    *                                                     *
    *   NOTE:   THIS SAMPLE PROGRAM DEMONSTRATES SOME OF  *
    *           THE NEW FEATURES FOR RELEASE 15.0+:       *
    *                                                     *
    *             PRTSIZE=161                             *
    *             INX INPUT FILE                          *
    *             OFF OUTPUT FILE                         *
    *             RPTSPCE                                 *
    *             MOVCVTX                                 *
    *             SEQUENCE NUMBERS OVER 4 DIGITS          *
    *             ADDRESSING FIELD OVER 4096 BYTES        *
    *               WITHOUT USING A PTR                   *
    *-------------------------------------------------------*
EQU PLANT-DEPT           INX1-5
EQU PLANT                INX1-2
EQU DEPARTMENT           INX3-5
EQU EMP-NR               INX6-9
EQU EMP-NAME             INX10-25
EQU BIRTH-DATE           INX26-31            /* YYMMDD
EQU EMPLY-DATE           INX32-37            /* YYMMDD
EQU EDUCATION            INX38-39
EQU HOURLY-RATE          INX46-49      2
EQU HOURS-WORKED         INX50-53      2
EQU PREV-YTD-GROSS       INX54-58-P    2C
EQU YTD-STATE-TAX        INX59-63-B    2C      /* 5 byte binary field
EQU YTD-FICA             INX64-69      2C
EQU OFF-RECORD           OFF001-6004
EQU OFF-RECORD                                 /* Redefines
EQU OFF-PLANT               (2)
EQU OFF-DEPT                (3)
EQU OFF-NR-EMP              (3)-P              /* Number employees
EQU OFF-HRS-WRKED           (6)-P     2C
EQU OFF-YTD-STATE           (6)-P     2C
EQU OFF-YTD-GROSS           (5)-B     2C      /* 5 byte binary field
EQU OFF-YTD-FICA            (19)      2C      /* 19 byte ebcdic field
EQU OFF-NET                 (10)-P    2C      /* 10 byte packed field
EQU OFF-MISC-DED            (4)-P     2C
EQU OFF-RESERVED         OFF5000               /* Set over 4096
```

```
EQU OFF-RESERVED
EQU OFF-RESVRD1              (7)-B     2C      /* Demonstrate MOVCVTX
EQU OFF-RESVRD2              (6)-P     2C      /* Demonstrate MOVCVTX
EQU OFF-RESVRD3              (7)       2C      /* Demonstrate MOVCVTX
EQU OFF-END-IT            OFF5999-6003-B       /* End of record
    TITLE1   'EXAMPLE 20 - QUARTERLY EARNINGS REPORT'
    TITLE2   'COMPUTER ASSOCIATES'
    REPORT  PLANT           SPACE4
            DEPARTMENT      (DEPT)
            EMP-NR          SPACE1
            HOURS-WORKED    (HRS WORKED)
            PREV-YTD-GROSS  SPACE1
            YTD-STATE-TAX   YTD-FICA
    BREAK 1 DEPARTMENT SB 0 SA 2               /* Level 1 control break field.
    BREAK 2 PLANT      SB 0 SA 2               /* Level 2 control break field.
NOTE:  INX WILL NOT GOTO EOJ AUTOMATICALLY; ONLY INF DOES
3000 GET INX ATEND 90000   /* Read an earnings record from input inx
    CHECKBREAKS ON BREAKS PERFORM 9000 THRU 11000
*  THE FOLLOWING ACCUMULATES AMT FIELDS &
*  SPECIFIES PRINT POSITIONS
*  ====> HOWEVER, "ACCUM" WITH "REPORT" WILL NOT
*  PRINT AT BREAK TIME
*        TO DO THAT, YOU NEED "HDR" STATEMENTS.
 ACCUM HOURS-WORKED   IN A 5 BYTE CTA, ON BREAKS PRINT IN POS 21 2
 ACCUM PREV-YTD-GROSS IN A 6 BYTE CTB, ON BREAKS PRINT IN POS 33 2C
 ACCUM YTD-STATE-TAX  IN A 6 BYTE CTC, ON BREAKS PRINT IN POS 48 2C
    ACCUM YTD-FICA       IN A 5 BYTE CTD, ON BREAKS PRINT IN POS 63 2C
    ACCUM ONE            IN A 3 BYTE CTH, ON BREAKS PRINT IN POS 122
    MOVE DEPARTMENT      TO WST1-3       /* Save for break time
    MOVE PLANT           TO WST11-14
    PRINT REPORT                         /* Print the detail line.
    GO TO 3000                           /* Go read next record.
*  THE FOLLOWING IS PERFORMED AT BREAK TOTAL TIME
9000IF @VAL-BREAK-LVL IS EQ C'1'          /* Chk for level 1 break.
       PERFORM 15000 THRU 16900           /*   EQ
       GOTO 11000.
10000
*   LEVEL 2 (OR POSSIBLY HIGHER) BREAK THEN
    MOVE CTH6-8-P  TO OFF-NR-EMP          /* Number of employees for dept
    MOVE WST1-3 TO PRT1                   /* Move dept nr from save area
    MOVE C'  PLANT  TOTAL' TO PRT6        /* Denote break totals.
    PERFORM 16200 THRU 16900
11000EXIT
15000
    MOVE WST11-14 TO PRT1
    MOVE C'DEPARTMENT TOTAL' TO PRT5
16200
BUILD OUTPUT QUARTERLY RECORD THAT IS 6004 BYTES
    MOVE SPACES    TO OFF-RECORD          /* Build output quarterly record
    MOVE WST11-14  TO OFF-PLANT
    MOVE WST1-3    TO OFF-DEPT
    MOVE CTH6-8-P  TO OFF-NR-EMP          /* Number of employees for dept
    MOVE CTA4-8-P  TO OFF-HRS-WRKED       /* Total hours worked
    MOVE CTB3-8-P  TO OFF-YTD-GROSS       /* Output is 5-byte binary
    MOVE CTC3-8-P  TO OFF-YTD-STATE
    MOVE CTD4-8-P  TO OFF-YTD-FICA        /* Output is 19-byte EBCDIC
* MOVCVTX CONVERTS MOVING OF X'..' FIELDS TO TARGET FIELD FORMAT
* FOLLOWING CODE DEMONSTRATES POWER OF MOVCVTX
    MOVE X'00'     TO OFF-NET             /* Move x'..'    to packed
* OFF-NET SHOULD HAVE X'00000000000000000000C' (10 BYTES PACKED)
    MOVE X'0C'     TO OFF-MISC-DED        /* Move x'..'    to packed
* OFF-MISC-DED SHOULD HAVE X'0000012C'
    MOVE X'0C'     TO OFF-RESVRD1         /* Move x'..'    to binary
* OFF-RESVRD1  SHOULD HAVE X'0000000000000C'
    MOVE X'0C'     TO OFF-RESVRD2         /* Move x'..'    to packed
```

```
* OFF-RESVRD2  SHOULD HAVE X'000000000012C'
    MOVE X'0C'     TO OFF-RESVRD3         /* Move x'..'    to EBCDIC
* OFF-RESVRD3  SHOULD HAVE X'F0F0F0F0F0F1F2'
    MOVE ZEROS     TO OFF-END-IT          /* Move bin zeros
    WRITE OFF                             /* Write record out
16900 EXIT
* 90000 PERFORM 10000 THRU 11000
90000 GOTO EOJ
99999999END
```

```
10/24/00           EXAMPLE 20 - QUARTERLY EARNINGS REPORT                PAGE    1
            COMPUTER ASSOCIATES, INFORMATION MANAGEMENT DIVISION
                                          PREV              YTD
                 EMP      HRS             YTD             STATE                 YTD
 PLANT    DEPT   NR.. .WORKED    .......GROSS  ............TAX  ........      FICA
  12      005   0002    40.00       23,711.60          1,104.68             506.31
  12      005   0008    41.00       32,523.04          1,515.19             694.46
  12      005   0014    40.05       30,967.20          1,442.70             661.24
  12      005   0017    45.00       38,417.28          1,789.79             820.31
  12      005   0026    40.00       38,372.40          1,787.70             819.36
  12      005   0103    40.00       35,185.92          1,639.25             751.32
  12      005   0105    43.00       45,628.00          2,125.72             974.29
  12      005   0129    51.00       27,601.20          1,285.89             589.36
  12      005   0155    48.00       30,548.32          1,423.19             652.29
  12      005   0158    42.00       37,953.52          1,768.18             810.41
  12      005   0176    40.00       33,435.60          1,557.70             713.94
  12      005   0194    38.00       39,195.20          1,826.03             836.93
  12      005   0340    40.00       34,662.32          1,614.85             740.14
  12  DEPARTMENT TOTAL 548.05      448,201.60         20,880.87            9,570.36  13
  12      008   0356    32.00       42,067.52          1,959.85             898.26
  12      008   0387    46.00       40,496.72          1,886.67             864.72
  12      008   0399    44.00       26,972.88          1,256.61             575.94
  12      008   0410    40.00       35,933.92          1,674.09             767.29
  12  DEPARTMENT TOTAL 162.00      145,471.04          6,777.22            3,106.21   4
  12      010   0423    42.00       49,218.40          2,292.99            1,050.95
  12      010   0452    45.00       26,853.20          1,251.04             573.39
  12      010   0578    40.00       40,092.80          1,867.85             856.09
  12      010   0594    40.00       32,657.68          1,521.46             697.33
  12  DEPARTMENT TOTAL 167.00      148,822.08          6,933.34            3,177.76   4
  12      015   0620    42.00       44,535.92          2,074.85             950.97
  12      015   0621    36.00       47,453.12          2,210.75            1,013.26
  12      015   0763    42.00       41,394.32          1,928.48             883.88
  12      015   0867    44.00       51,791.52          2,412.87            1,105.90
  12  DEPARTMENT TOTAL 164.00      185,174.88          8,626.95            3,954.01   4
  12      120   1034    54.00       60,662.80          2,826.17            1,295.32
  12  DEPARTMENT TOTAL  54.00       60,662.80          2,826.17            1,295.32   1
 120    PLANT  TOTAL  1095.05      988,332.40         46,044.55           21,103.66  26
```

The report has been modified to fit onto this page.

Example 21

ACCUM, BREAK, and CHECKBREAKS with Total Time Calculations, Multiple HDR

This example shows a comparison of the sales history file for each salesmen with previous year's sales dollars, this year's sales dollars, amount of increased sales dollars, the percent of increase, and a grand total.

This example shows how easily this can be accomplished with VISION:Report using BREAK and CHECKBREAK for control fields comparisons and ACCUM for adding sales dollars. When a break does occur, control is passed to Statements 200-300, which subtract the previous year's amount from this year's amount to give the increase amount. The percent of increase is computed by dividing previous year's amount into the increase amount giving the percent of increase. The increase amount and increase percent are stored in CTC and CTD, and are automatically printed by VISION:Report in each level 1 break and also on the final total level.

VISION:Report Statements

```
INFCARD                                    /* VSE only
EQU P-SALES-NR    PRT2
EQU INCR-SALES    WST1-6-P
EQU SALESMAN      INF1-4
EQU PREV-YR-SALES  INF36-40-P
EQU THIS-YR-SALES INF41-45-P
004 HDR 1A 1DATE ENDING  $IPLDAT$         SALES COMPARISONS
005 HDR 1B          PAGE$PG$
007 HDR 2A 0SALES    SALES $       SALES $          SALES $
008 HDR 2B            %
010 HDR 3A   MAN     LAST YEAR   THIS YEAR       INCREASE
011 HDR 3B INCR.
020 *
025 BREAK 1 SALESMAN SB 1 SA 1             /* Define control break field.
030 *
050 GET                                    /* Read a record.
060 CHECKBREAKS ON BREAKS PERFORM 200 THRU 300    /* Do break compare.
070*
100 MOVE SALESMAN TO P-SALES-NR            /* Move salesman nr to print.
110 ACCUM PREV-YR-SALES    IN A 6 BYTE CTA, ON BREAKS PRINT IN POS 10 2C
120 ACCUM THIS-YR-SALES    IN A 6 BYTE CTB, ON BREAKS PRINT IN POS 30 2C
130 ACCUM NONE             IN A 6 BYTE CTC, ON BREAKS PRINT IN POS 50 2C
140 ACCUM NONE             IN A 4 BYTE CTD, ON BREAKS PRINT IN POS 74 1
150 GOTO 050
170 *
180 *  The following will be done at total break-time.
190 *
200 MOVE CTB3-8-P TO CTC3-8-P              /* Move this-yr sales to CTC.
210 SUB CTA3-8-P FR CTC3-8-P               /* Sub prev-yr sales from it gives incr.
220 DIVD CTC3-8-P 0D BY CTA3-8-P 2D GIVING CTD6-8-P 1DR   /* Calc incr. %
300 EXIT                     /* Give control to VISION:Report for printing.
9999END
```

```
DATE ENDING 01/15/00        SALES COMPARISONS        PAGE  1

SALES        SALES $        SALES $      SALES $      %
 MAN        LAST YEAR      THIS YEAR    INCREASE     INCR.

1001        686,420.66      742,658.15   56,237.49     8.2

1124        506,936.70      480,054.54   26,882.16-    5.3-

1256        430,555.14      568,432.67  137,877.53    32.0

1280        122,697.40      446,334.36  323,636.96   263.8

1432        644,108.41      722,963.06   78,854.65    12.2

1549        833,468.38      746,772.04   86,696.34-   10.4-

1573               .00      345,127.22  345,127.22      .0

1610        710,325.55      764,588.85   54,263.30     7.6

          3,934,512.24    4,816,930.89  882,418.65    22.4
```

Example 22

Amortization Schedule, Calculations, No Input/Output Files, LINECOUNT, DISPLAY, ACCEPT, Arithmetic Operations, Multiple HDR, PERFORM/THRU

This example computes and prints amoritization schedules, like the one shown below, using the statements shown on the following page. This example shows the remaining balance for the last three years of a 30-year mortgage.

```
09/19/02                  AMORTIZATION SCHEDULE                    PAGE    1
BEGINNING BALANCE             INTEREST RATE                  MONTHLY PAYMENT
-------------------------------------------------------------------------------
 DATE    PAY-NR       PAYMENT       INTEREST      PRINCIPAL     END BALANCE
CURR. BALANCE:   200,000.00  INTEREST RATE:  6.250   MONTHLY PAYMENT:  3,000.00
09/19/02                  AMORTIZATION SCHEDULE                    PAGE    1
BEGINNING BALANCE             INTEREST RATE                  MONTHLY PAYMENT
-------------------------------------------------------------------------------
 DATE    PAY-NR       PAYMENT       INTEREST      PRINCIPAL     END BALANCE
01  2007  53        3,000.00         434.32       2,565.68      80,823.40
02  2007  54        3,000.00         420.96       2,579.04      78,244.36
03  2007  55        3,000.00         407.52       2,592.48      75,651.88
04  2007  56        3,000.00         394.02       2,605.98      73,045.90
05  2007  57        3,000.00         380.45       2,619.55      70,426.35
06  2007  58        3,000.00         366.80       2,633.20      67,793.15
07  2007  59        3,000.00         353.09       2,646.91      65,146.24
08  2007  60        3,000.00         339.30       2,660.70      62,485.54
09  2007  61        3,000.00         325.45       2,674.55      59,810.99
10  2007  62        3,000.00         311.52       2,688.48      57,122.51
11  2007  63        3,000.00         297.51       2,702.49      54,420.02
12  2007  64        3,000.00         283.44       2,716.56      51,703.46
YEARLY TOTAL       36,000.00       4,314.38      31,685.62
01  2008  65        3,000.00         269.29       2,730.71      48,972.75
02  2008  66        3,000.00         255.07       2,744.93      46,227.82
03  2008  67        3,000.00         240.77       2,759.23      43,468.59
04  2008  68        3,000.00         226.40       2,773.60      40,694.99
05  2008  69        3,000.00         211.95       2,788.05      37,906.94
06  2008  70        3,000.00         197.43       2,802.57      35,104.37
07  2008  71        3,000.00         182.84       2,817.16      32,287.21
08  2008  72        3,000.00         168.16       2,831.84      29,455.37
09  2008  73        3,000.00         153.41       2,846.59      26,608.78
10  2008  74        3,000.00         138.59       2,861.41      23,747.37
11  2008  75        3,000.00         123.68       2,876.32      20,871.05
12  2008  76        3,000.00         108.70       2,891.30      17,979.75
YEARLY TOTAL       36,000.00       2,276.29      33,723.71
01  2009  77        3,000.00          93.64       2,906.36      15,073.39
02  2009  78        3,000.00          78.51       2,921.49      12,151.90
03  2009  79        3,000.00          63.29       2,936.71       9,215.19
04  2009  80        3,000.00          48.00       2,952.00       6,263.19
05  2009  81        3,000.00          32.62       2,967.38       3,295.81
06  2009  82        3,000.00          17.17       2,982.83         312.98
07  2009  83          314.61           1.63         312.98
YEARLY TOTAL       18,314.61         334.86      17,979.75
GRAND TOTAL       246,314.61      46,314.61     200,000.00
```

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP22:    AMORTIZATION SCHEDULE.                        *
*                                                          *
*            CALCULATIONS, NO INPUT/OUTPUT FILES,          *
*            LINECOUNT, ARITHMETIC OPERATIONS,             *
*            MULTIPLE HDR, PERFORM.                        *
*                                                          *
*  SHOW LAST 3 YEARS OF MORTGAGE SCHEDULE.                 *
*                                                          *
*  CARD INPUT POSITIONS ARE AS FOLLOWS:       (EXAMPLES)   *
*  NO SPECIFIC CARD COLUMN FORMAT REQUIRED                 *
*  MONTHLY PAYMENT    $540.00  ENTER M=540.00              *
*  PRINCIPAL        $32,000.00 ENTER P=32000.00            *
*  INTEREST           18%      ENTER I=18.00               *
*  NUMBER OF PAYMENTS 3 YEARS  ENTER N=36                  *
*  DATE OTHER THAN THIS MONTH  ENTER D=061980              *
*  NO SCHEDULE WANTED          ENTER S=NO                  *
*  FULL SCHEDULE TO PRINT NO ENTRY MADE                    *
*                                                          *
* EXAMPLE OF INPUT CARD:                                   *
*                                                          *
*     M=540.00 P=32000.00 I=10.00 N=60 S=NO                *
*                                                          *
************************************************************
OPTION NOSEQ                   /* NO NEED TO HAVE SEQUENCE CHECKING
OPTION BWZ=YES
OPTION STMTS=400,GENSIZE=12000 /* GENSIZE LEFT IN FOR COMPATIBILITY
OPTION LITSIZE=4000            /* LITSIZE LEFT IN FOR COMPATIBILITY
OPTION STMTEND=80              /* LET IT END ON COLUMN 80

*INFCARD                   /* IF VSE, REMOVE * AT POSITION 1

EQU MM          WST1-2
EQU YEAR        WST3-6
EQU COUNTER     WST7-9
EQU ONE         WST11
EQU MO-PAY-AMT  WST12-17-P
EQU INT-RATE-%  WST21-23-P
EQU CURR-BAL    WST24-29-P
EQU FN-INT-PAID WST30-35-P
EQU YR-INT-PAID WST36-41-P
EQU MO-INT-AMT  WST42-47-P
EQU YR-PAY-AMT  WST48-53-P
EQU FN-PAY-AMT  WST54-59-P
EQU CALC-INT    WST60-65-P
EQU PRIN-AMT    WST66-71-P
EQU PAY-NR      WST75-77-P
EQU IN-BALANCE  WST78-87
EQU IN-INT-RT   WST88-92
EQU IN-MON-PAY  WST93-99
EQU I-12        WST100-105-P
EQU ST7         WST106-110-P
EQU ST8         WST111-117-P
EQU ST11        WST118-122-P
EQU IN-DATE     WST123-128
EQU #PMT        WST129-131
EQU FLAG        WST132
EQU COL-COUNT   WST133-134
EQU SCHEDULE    WST135-136
EQU PRINCIPAL-SW WST138
EQU MO-PAY-SW    WST139
EQU SCHEDULE-SW  WST140
```

```
EQU DATE-SW      WST141
EQU NR-PAY-SW    WST142
EQU INTEREST-SW  WST143
EQU SWITCH-AREA  WST138-143
EQU MAX-SIZE     WST144-145
EQU CHAR-CNT     WST146-147
EQU CHAR-CNT-$   WST148-149
EQU CHAR-CNT-DEC WST150
EQU DEC-FLAG     WST151

    HDR 1A 1$IPLDAT$                  AMORTIZATION SCHEDULE
    HDR 1B  PAGE $PG$
    HDR 2A 0BEGINNING BALANCE             INTEREST RATE                 MONTHL
    HDR 2B Y PAYMENT
    HDR 3A
    HDR 4A  ------------------------------------------------------------------
    HDR 4B ----------
    HDR 5A   DATE    PAY-NR       PAYMENT       INTEREST      PRINCIPAL     EN
    HDR 5B D BALANCE
    LINECOUNT 60
050 EXIT                          /* THIS STMT USED FOR TRANSFER POINT
    SET PTD HDC1                  /* SET INDEX POINTER TO HDR-C FOR EDITING
    MOVE SPACES TO PTD1-132
    MOVE ZEROS TO WST12-17-P      /* ZERO WORKING STORAGE FOR COUNTER USE
    MOVE WST12-77 TO WST18        /* SPREAD CNTR ACROSS WST FOR 11 MORE
    MOVE ZEROES TO WST123-137
    MOVE ZEROES TO WST78-99
    MOVE ZEROES TO WST144-150
    MOVE ZEROS TO I-12
    MOVE ZEROS TO ST7
    MOVE ZEROS TO ST8
    MOVE ZEROS TO COUNTER
    MOVE C'1' TO ONE
    MOVE C'12' TO ST11
    MOVE BLANKS TO SWITCH-AREA
    MOVE BLANKS TO IN-DATE
    MOVE BLANKS TO DEC-FLAG
    *
100 GET INF ATEND EOJ             /* READ INPUT CARD, ATEND GOTO EOJ
    SET PTA INF1
101 IF PTA1 IS EQ C'P'             /* STEPPING ACROSS INPUT CARD
      GOTO 105.
    IF PTA1 IS EQ C'M'            /* MONTHLY PAYMENT
      GOTO 108.
    IF PTA1 IS EQ C'I'            /* INTEREST?
      GOTO 110.
    IF PTA1 IS EQ C'N'            /* NUMBER OF PAYMENTS
      GOTO 113.
    IF PTA1 IS EQ C'D'            /* DATE ENTERED?
      GOTO 116.
    IF PTA1 IS EQ C'S'            /* FULL SCHEDULE REQUIRED?
      GOTO 120.
    IF COL-COUNT IS GT C'79'      /* END OF CARD CHECK
      MOVE ZEROES TO COL-COUNT
      GOTO 140.
    SET PTA UP 1                  /* KEEP LOOKING
    ADD C'1' TO COL-COUNT         /* COUNT COLUMNS
    GOTO 101
105 IF PRINCIPAL-SW IS EQ TO C'P' /* THERE IS ALREADY A PRINCIPAL ENTERED
      PERFORM 901 THRU 909        /* ERROR MESSAGE
      GOTO 050.                   /* GET ANOTHER INPUT, THIS WON'T WORK
    MOVE C'P' TO PRINCIPAL-SW
    SET PTA UP 2                  /* MOVE ACROSS EQUAL SIGN TO FIRST DIGIT
    ADD C'2' TO COL-COUNT         /* COUNT COLUMNS OF CARD
    SET PTB PTA1
```

```
    PERFORM 925 THRU 950          /* ROUTINE TO COUNT DIGITS IN INPUT
    MOVE C'10' TO MAX-SIZE        /* MAXIMUM OF 10 DIGITS ALLOWED IN PRINCIPAL
    IF CHAR-CNT-$ IS GT C'08'
      GOTO 921.
    SUB CHAR-CNT FR MAX-SIZE      /* HOW MANY CHAR IN INPUT?
    IF DEC-FLAG IS NOT EQ C'Y'
      SUB C'2' FR MAX-SIZE.       /* ALLOW FOR 2 DECIMAL PLACES.
107 SET PTC IN-BALANCE            /* POINT TO WORK AREA
    MOVE ZEROS TO IN-BALANCE      /* FILL WITH ZEROES
    PERFORM 960 THRU 975          /* ROUTINE TO PLACE INPUT INTO WORK AREA
    SET PTA PTB1                  /* MOVE TO NEXT  COLUMN SEARCHING FOR CODES
    GOTO 101
108 IF MO-PAY-SW IS EQ C'M'       /* THERE IS ALREADY A MONTHLY PAYMENT
      PERFORM 901 THRU 909        /* ERROR MESSAGE
      GOTO 050.                   /* GET ANOTHER INPUT, THIS WON'T WORK
    MOVE C'M' TO MO-PAY-SW
    SET PTA UP 2                  /* MOVE ACROSS EQUAL SIGN TO FIRST DIGIT
    ADD C'2' TO COL-COUNT         /* COUNT COLUMNS OF CARD
    SET PTB PTA1
    PERFORM 925 THRU 950          /* ROUTINE TO COUNT DIGITS IN INPUT
    MOVE C'07' TO MAX-SIZE        /* MAXIMUM OF 7 DIGITS FOR MONTH PAYMT
    IF CHAR-CNT-$ IS GT C'05'
      GOTO 921.                   /* ERROR MESSAGE
    IF CHAR-CNT IS GT MAX-SIZE
      GOTO 921.
    SUB CHAR-CNT FR MAX-SIZE      /* HOW MANY CHAR IN INPUT?
    IF DEC-FLAG IS NOT EQ C'Y'
      SUB C'2' FR MAX-SIZE.       /* ALLOW FOR 2 DECIMAL PLACES.
109 SET PTC IN-MON-PAY            /* POINT TO MONTHLY PAYMENT WORK AREA
    MOVE ZEROES TO IN-MON-PAY     /* FILL-IN WITH ZEROES
    PERFORM 960 THRU 975
    SET PTA PTB1                  /* MOVE TO NEXT  COLUMN SEARCHING FOR CODES
    GOTO 101
110 IF INTEREST-SW IS EQ C'I'     /* ALREADY AN INTEREST AMT ENTERED
      PERFORM 901 THRU 909        /* ERROR MESSAGE
      GOTO 050.                   /* GET ANOTHER INPUT, THIS WON'T WORK
    MOVE C'I' TO INTEREST-SW
    SET PTA UP 2                  /* MOVE ACROSS EQUAL SIGN TO FIRST DIGIT
    ADD C'2' TO COL-COUNT         /* COUNT COLUMNS OF CARD
    SET PTB PTA1
    PERFORM 925 THRU 950          /* ROUTINE TO COUNT DIGITS IN INPUT
    MOVE C'05' TO MAX-SIZE        /* MAXIMUM OF 5 DIGITS FOR INTEREST
    IF CHAR-CNT IS GT MAX-SIZE
      GOTO 921.
    SUB CHAR-CNT FR MAX-SIZE      /* HOW MANY CHAR IN INPUT?
    SET PTC IN-INT-RT
    IF CHAR-CNT-$ IS GT C'02'
      GOTO 921.
    IF CHAR-CNT-$ IS LT C'02'
      SET PTC UP 1.
    MOVE ZEROES TO IN-INT-RT      /* FILL IN WITH ZEROES
    PERFORM 970 THRU 975
    SET PTA PTB1                  /* MOVE TO NEXT  COLUMN SEARCHING FOR CODES
    GOTO 101
113 IF NR-PAY-SW IS EQ TO C'N'    /* THERE ALREADY IS NUMBER OF PAYMT ENTERED
      PERFORM 901 THRU 909        /* ERROR MESSAGE
      GOTO 050.                   /* GET ANOTHER INPUT, THIS WON'T WORK
    MOVE C'N' TO NR-PAY-SW
    SET PTA UP 2                  /* MOVE ACROSS EQUAL SIGN TO FIRST DIGIT
    ADD C'2' TO COL-COUNT         /* COUNT COLUMNS OF CARD
    SET PTB PTA1
    PERFORM 925 THRU 950          /* ROUTINE TO COUNT DIGITS IN INPUT
    MOVE C'03' TO MAX-SIZE        /* ONLY 3 DIGITS ALLOWED
    IF CHAR-CNT IS GT MAX-SIZE
      GOTO 921.
```

```
    SUB CHAR-CNT FR MAX-SIZE       /* HOW MANY CHAR IN INPUT?
    SET PTC #PMT                   /* POINT TO NR OF PAYMENTS WORK AREA
    MOVE ZEROES TO #PMT
    PERFORM 960 THRU 975
    SET PTA PTB1                   /* MOVE TO NEXT  COLUMN SEARCHING FOR CODES
    GOTO 101
116 IF DATE-SW IS EQ TO C'D'       /* ALREADY A DATE ENTERED
      PERFORM 901 THRU 909         /* ERROR MESSAGE
      GOTO 050.                    /* GET ANOTHER INPUT, THIS WON'T WORK
    MOVE C'D' TO DATE-SW
    SET PTA UP 2                   /* MOVE ACROSS EQUAL SIGN TO FIRST DIGIT
    ADD C'2' TO COL-COUNT          /* COUNT COLUMNS OF CARD
    SET PTB PTA1
    PERFORM 925 THRU 950           /* ROUTINE TO COUNT DIGITS IN INPUT
    MOVE C'06' TO MAX-SIZE
    IF CHAR-CNT IS GT MAX-SIZE
      GOTO 921.
    SUB CHAR-CNT FR MAX-SIZE       /* HOW MANY CHAR IN INPUT?
    SET PTC IN-DATE
    MOVE ZEROES TO IN-DATE
    PERFORM 960 THRU 975
    SET PTA PTB1                   /* MOVE TO NEXT  COLUMN SEARCHING FOR CODES
    GOTO 101
120 IF SCHEDULE-SW IS NOT BLANK
      PERFORM 901 THRU 909         /* ERROR MESSAGE
      GOTO 050.                    /* GET ANOTHER INPUT, THIS WON'T WORK
    MOVE C'N' TO SCHEDULE-SW
    SET PTA UP 4                   /* MOVE ACROSS EQUAL SIGN TO FIRST DIGIT
    ADD C'4' TO COL-COUNT          /* COUNT COLUMNS OF CARD
    SET PTB PTA1
    GOTO 101
140 IF IN-DATE IS BLANK
      GOTO 141.
    IF IN-DATE IS NUMERIC          /* TEST AND USE START DATE IF NUMERIC
      MOVE IN-DATE TO WST1-6
      GOTO 142.
141 MOVE VAL5-6 TO MM              /* USE CURR MON FROM VAL-AREA FOR 1ST PAY.
    MOVE C'20' TO WST3-4           /* MAKE 20XX YEAR
    MOVE VAL11-12 TO WST5-6        /* USE CURR YEAR
142 IF IN-BALANCE IS NOT NUMERIC   /* TEST IF AMOUNT IS NUMERIC ?
      PRINTHEX IN-BALANCE
      PERFORM 910 THRU 920         /* PRINT BAD REC IF NOT NUMERIC
      GOTO 050.                    /* AND GET ANOTHER RECORD
    MOVE IN-BALANCE TO CURR-BAL    /* USE AS CURR. MORTGAGE BALANCE
    MOVE CURR-BAL TO PRT14-24 2C       /* MOVE TO PRINT HDR
    MOVE C'CURR. BALANCE: '  TO PRT1
    *
    IF IN-INT-RT IS ZEROES OR
      IF IN-INT-RT  IS NOT NUMERIC /* TEST IF RATE IS NUMERIC ?
      GOTO 900.
    MOVE IN-INT-RT  TO INT-RATE-%  /* SAVE RATE
    MOVE INT-RATE-% TO PRT45-50 3     /* MOVE TO PRINT HDR
    MOVE C'INTEREST RATE: '  TO PRT30
    *
    IF IN-MON-PAY IS NOT NUMERIC   /*TEST IF MONTHLY PAY IS NUMERIC ?
      PERFORM 910 THRU 920         /* PRINT BAD RECORD IF NOT NUMERIC
      GOTO 050.                    /* AND GET ANOTHER RECORD
    MOVE IN-MON-PAY TO MO-PAY-AMT  /* SAVE MONTH AMT
    MOVE MO-PAY-AMT TO PRT66-72  2C    /* MOVE TO PRINT HDR
    MOVE C'MONTHLY PAYMENT: '  TO PRT54
    PRINT                          /* SHOW INPUT DATA
    *
160 IF CURR-BAL IS GT P'0'
      IF MO-PAY-AMT IS GT P'0'
      GOTO 170.                    /* ALL INFORMATION FOR FULL AMORTIZATION
```

```
    IF MO-PAY-AMT IS LT P'1' OR    /* FIGURE MONTHLY PMT OR BALANCE ONLY
      IF CURR-BAL IS LT P'1'
      IF SCHEDULE-SW IS EQ C'N'      /* NO SCHEDULE INDICATED
        GOTO 599.
    IF CURR-BAL LT P'1'            /* ROUTINE TO FIGURE BALANCE
      PERFORM 600 THRU 700
      PERFORM 800 THRU 850
      GOTO 170.
    IF MO-PAY-AMT IS LT P'1'       /* ROUTINE TO FIGURE MONTHLY PAYMENTS
      PERFORM 600 THRU 700
      PERFORM 770 THRU 799
      GOTO 170.
    PERFORM 901 THRU 909
    GOTO 050                       /* MUST BE AN ERROR START ALL OVER
*  AMORTIZATION SCHEDULE
170 DOHEADERS PAGEONE              /* FORCE NEW PAGE & HEADERS
    *
200 ADD C'1' TO PAY-NR             /* INCREMENT PAYMENT NUMBER
    MULT CURR-BAL 2D BY INT-RATE-% 5D GIVING CALC-INT 2D /* CALC INT. ANNUAL
    DIVD CALC-INT 2D BY C'12' 0D GIVING MO-INT-AMT 2DR /* CALC MO. INT.
    ADD MO-INT-AMT TO YR-INT-PAID  /* ACCUM INT. FOR YEAR
    ADD MO-INT-AMT TO FN-INT-PAID  /*              & FINAL
    MOVE MO-PAY-AMT TO PRIN-AMT    /* CALC PRINCIPAL
    SUB MO-INT-AMT FR PRIN-AMT     /*      MONTHLY AMOUNT
    IF PRIN-AMT IS GT CURR-BAL     /* CK FOR
      MOVE CURR-BAL TO PRIN-AMT    /*     END OF
      MOVE CURR-BAL TO MO-PAY-AMT  /*         SCHEDULE PAYMENTS
      ADD MO-INT-AMT TO MO-PAY-AMT. /*           AMD COMPUTATION.
    SUB PRIN-AMT FR CURR-BAL       /* CALC NEW CURR/ENDING BALANCE
    ADD MO-PAY-AMT TO YR-PAY-AMT   /* ACCUM TOTAL PAID FOR YEAR
    ADD MO-PAY-AMT TO FN-PAY-AMT   /*                    & FINAL
    *
    MOVE CURR-BAL TO PRT60 2C      /* CURR. BALANCE TO PRINT
    MOVE PRIN-AMT TO PRT45 2C      /* PRINCIPAL
    MOVE MO-INT-AMT TO PRT30 2C    /* INTEREST
    MOVE MO-PAY-AMT TO PRT15 2C    /* PAYMENT
    MOVE PAY-NR TO PRT8 0          /* PAYMENT NUMBER
    MOVE YEAR TO PRT5              /* YEAR
    MOVE MM TO PRT1                /* MONTH
    IF MO-INT-AMT IS GT MO-PAY-AMT /* CK IF THIS THING CAN COMPUTE
      PRINT
      MOVE C'PAYMENT TOO SMALL TO COMPUTE' TO PRT1
      DISPLAY PRT1-40              /* TYPE IT ON CONSOLE
      PRINT DOUBLESPACED           /* PRINT BUMMER MESSAGE
      GOTO 050.                    /* GO BACK TO START
    PRINT                          /* PRINT A DETAIL LINE
    IF CURR-BAL IS ZERO            /* TEST FOR END OF THIS SCH.
      GOTO 400.                    /* EQ, GOTO FINAL TOTALS
    *
    ADD C'1' TO MM                 /* INCREMENT MONTH
    IF MM IS EQ TO C'13'           /* TEST FOR BEGIN OF NEW YEAR
      ADD C'1' TO YEAR             /* BUMP YR BY 1
      MOVE C'01' TO MM             /* MAKE MONTH 01
300  MOVE YR-PAY-AMT TO PRIN-AMT   /* CALC THE YR AMT
      SUB YR-INT-PAID FR PRIN-AMT  /*      FOR PRIN PAID
      MOVE PRIN-AMT TO PRT45 2C    /* MOVE YR PRIN PAID TO PRINT
      MOVE YR-INT-PAID TO PRT30 2C /* INTEREST
      MOVE YR-PAY-AMT TO PRT15 2C  /* PAYMENTS
      MOVE C'YEARLY TOTAL' TO PRT1 /* TOTAL LEVEL
      PRINT                        /* PRINT YR TOTALS
      PRINT                        /* BLANK AN EXTRA LINE
      MOVE ZEROS TO YR-PAY-AMT     /* ZERO YR PAY
      MOVE ZEROS TO YR-INT-PAID.   /*       & INT
350 EXIT                           /* EXIT HERE ON FINAL TOTAL LEVEL
    *
```

```
    GOTO 200                      /* GO COMPUTE NEXT PAYMENT NUMBER
    *   END OF SCHEDULE - PRINT YEARLY & FINAL TOTAL RESULTS
400 PERFORM 300 THRU 350          /* GO PRINT YEARLY TOTALS
    MOVE FN-PAY-AMT TO PRIN-AMT   /* CALC THE GRAND TOTAL AMT
    SUB FN-INT-PAID FR PRIN-AMT   /*      FOR PRIN PAID
    MOVE PRIN-AMT TO PRT45 2C     /* MOVE PRIN TO PRINT
    MOVE FN-INT-PAID TO PRT30 2C  /* INTEREST
    MOVE FN-PAY-AMT TO PRT15 2C   /* PAYMENTS
    MOVE C'GRAND TOTAL' TO PRT1   /* TOTAL LEVEL
    PRINT DOUBLESPACED            /* PRINT GRAND TOTALS
    GOTO 050                      /* GO PROCESS NEXT SCHEDULE.
    *
*  FIND MONTHLY PAYMENT OR BEGINNING BALANCE
599 DOHEADERS PAGEONE             /* PRINT HEADLINES FOR REPORT
*
600 IF INT-RATE-% IS ZEROS OR     /* CHECK FOR VALID INTEREST RATE
       IF INT-RATE-% IS NOT NUMERIC
       GOTO 900.
    MOVE #PMT TO COUNTER
    DIVD INT-RATE-% 5D BY ST11 0D GIVING I-12 8D    /* PRELIMINARY CALCULATIONS
    MOVE I-12 TO ST7                                 /*    IN ORDER TO COMPUTE
    ADD C'100000000' TO ST7                          /*    MONTHLY PAYMENT OR
    MOVE ST7 TO ST8                                  /*    BEGINNING BALANCE
    SUB  C'1' FROM COUNTER
625 IF COUNTER IS ZERO
       GOTO 650.
    MULT ST7 8D BY ST8 8D GIVING ST8 8D
    SUB C'1' FR COUNTER
    GOTO 625
650 DIVD ONE  0D BY ST8 8D GIVING ST8 8D
    MOVE C'100000000' TO ST11
    SUB ST8 FR ST11
700 EXIT
750 IF MO-PAY-AMT IS ZEROS        /* COMPUTE MONTHLY PAYMENT
770 DIVD I-12 8D BY ST11 8D GIVING ST8 8D
    MULT CURR-BAL 2D BY ST8 8D GIVING MO-PAY-AMT 2DR
    MOVE MO-PAY-AMT TO  PTD61 2C
799 EXIT
    MOVE INT-RATE-% TO PTD33 2    /* SET UP THE PRINT LINE
    MOVE MM TO PRT1
    MOVE YEAR TO PRT5
    MOVE #PMT TO PRT11 0
    MOVE MO-PAY-AMT TO PRT15 2C
    MOVE CURR-BAL TO PRT45 2C
    PRINT DOUBLESPACED
    GOTO 050.
800 DIVD ST11 8D BY I-12 8D GIVING ST8 4D          /* CALCULATE BEG. BALANCE
    MULT MO-PAY-AMT 2D BY ST8 4D GIVING CURR-BAL 2DR
    MOVE CURR-BAL TO PTD1 2C
850 EXIT
    MOVE INT-RATE-% TO PTD33 2    /* SET UP PRINT LINE
    MOVE MM TO PRT1
    MOVE YEAR TO PRT5
    MOVE #PMT TO PRT11 0
    MOVE MO-PAY-AMT TO PRT15 2C
    MOVE CURR-BAL TO PRT45 2C
    PRINT
    GOTO 050
900 MOVE C'ERROR-MUST ENTER INTEREST' TO PRT5
    PRINT
    MOVE INF1-80 TO PRT1
    PRINT
    GOTO 050
901 MOVE INF1-80 TO PRT1
    PRINT
```

```
    MOVE C'ERROR IN CODES, M,P,I,N,S,D' TO PRT1
    PRINT
909 EXIT
910 MOVE INF1-80 TO PRT1           /* MOVE BAD RECORD TO PRINT
    MOVE C'DATA IN POS. 1-80 NOT NUMERIC.' TO PRT30
    DOHEADERS
    PRINT DOUBLESPACED
920 EXIT
921 MOVE INF1-80 TO PRT1
    PRINT
    MOVE C'NUMBERS IN INPUT EXCEED MAXIMUM' TO PRT1
    PRINT
    GOTO 050
925 MOVE C'N' TO DEC-FLAG          /* CLEAR
    MOVE ZEROES TO CHAR-CNT-$
    MOVE ZEROES TO CHAR-CNT-DEC
    MOVE ZEROES TO CHAR-CNT
935 IF PTB1 IS EQ TO C' '          /* FINISHED WITH THIS NUMBER ?
    ADD CHAR-CNT-$ TO CHAR-CNT
    ADD CHAR-CNT-DEC TO CHAR-CNT
    ADD CHAR-CNT TO COL-COUNT      /* COUNT COLUMNS
      GOTO 950.
    IF PTB1 IS EQ C'.'             /* IS IT A DECIMAL POINT?
      MOVE C'Y' TO DEC-FLAG        /* TURN ON FLAG TO SAY SO
      SET PTB UP 1                 /* PASS IT UP
      GOTO 945.
    ADD C'1' TO CHAR-CNT-$         /* COUNT NUMBER OF DIGITS BEFORE DECIMAL
    SET PTB UP 1                   /* POINT TO NEXT NUMBER
    GOTO 935
945 IF PTB1 IS BLANK
      GOTO 935.
    IF CHAR-CNT-DEC IS GT C'2'
      GOTO 921.
    ADD C'1' TO CHAR-CNT-DEC       /* COUNT DIGITS AFTER DECIMAL
    SET PTB UP 1
    GOTO 945
950 EXIT
960 IF MAX-SIZE IS NOT ZEROES
      SET PTC UP 1
      SUB C'1' FR MAX-SIZE
      GOTO 960.
970 IF PTA1 IS EQ C'.'             /* DON'T MOVE IN A DECIMAL POINT
      SET PTA UP 1
      GOTO 970.
    MOVE PTA1 TO PTC1              /* MOVE IN INPUT NUMBER, 1 ATA TIME
    SET PTA UP 1
    SET PTC UP 1
    SUB C'1' FR CHAR-CNT
    IF CHAR-CNT IS NOT ZEROES      /* FINISHED MOVING THIS NUMBER?
      GOTO 970.                    /*  NO
975 EXIT
9999END
```

```
M=3000.00 P=200000.00 I=6.25 N=60 S=NO
/*  *                                                          *
/*  *  CARD INPUT POSITIONS ARE AS FOLLOWS:       (EXAMPLES)  *
/*  *  NO SPECIFIC CARD COLUMN FORMAT REQUIRED                 *
/*  *                                                          *
/*  *  MONTHLY PAYMENT    $3000.00 ENTER M=3000.00            3
/*  *  PRINCIPAL       $100,000.00 ENTER P=100000.00          *
/*  *  INTEREST           6.5      ENTER I=06.50              *
/*  *  NUMBER OF PAYMENTS 3 YEARS  ENTER N=36                 *
/*  *  DATE OTHER THAN THIS MONTH  ENTER D=061990             *
/*  *  NO SCHEDULE WANTED          ENTER S=NO                 *
/*  *  FULL SCHEDULE TO PRINT NR ENTRY MADE                   *
/*  *                                                          *
/*  * EXAMPLE OF INPUT CARD:                                   *
/*  *                                                          *
/*  *     M=3000.00 P=100000.00 I=10.00 N=60 S=NO              *
/*  *                                                          *
/*  ************************************************************
```

Example 23

Match Records of a Transaction File Against a Master File and Create a New Master File

A common requirement is the matching of transaction records to a master file. As an example, take timekeeping records (DET FILE) and match them to a payroll master file (INF) by employee number. On matching conditions, generate an output record (OFA) consisting of the timekeeping record and the employee name, paygrade, and hourly rate taken from the matching payroll master record.

Any unmatched timekeeping transactions are to be printed and dropped. The assumption is made that the payroll master file is VSAM KSDS with employee number as the key, and the transactions reside on a sequential disk file in employee number sequence. The output records will be written to a disk file.

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP23:    MATCH MASTER (INF SEQ)  AGAINST               *
*            TRANSACTION INPUT FILES (DET SEQ) AND         *
*            CREATE NEW MASTER FILE (OFA SEQ).             *
*                                                          *
************************************************************
*INFDISC11000110SSYS005            /* IF VSE, REMOVE * AT POSITION 1
*DETDISC80000080SSYS009            /* IF VSE, REMOVE * AT POSITION 1
*OFADISC11000110SSYS006            /* IF VSE, REMOVE * AT POSITION 1
    HDR 1A 1$IPLDAT$        UNMATCHED TIMEKEEPING TRANSACTIONS
*
10  GET                           /* READ PAYROLL MASTER FILE RECORD.
*
20  GET DET                       /* READ TIMEKEEPING RECORD.
*
30  IF VAL196-197 IS EQ TO C'EE'  /* TEST BOTH FILES.
        GOTO EOJ.                 /* FOR EOF, STOP WHEN EQUAL.
*
    IF INF1-6 IS LT DET1-6        /* COMP EMP.NR. IN INF/DET RECORDS.
        MOVE INF1-110 TO OFA1     /* WRITE OUT OFA RECORD
        WRITE OFA                 /*
        GET                       /* READ PAYROLL MASTER FILE RECORD.
        GOTO 30.                  /* GO COMPARE NEXT RECORD.
*
    IF INF1-6 IS GT DET1-6        /* COMP EMP.NR. IN INF/DET RECORDS.
        MOVE DET1-15 TO PRT1      /* DET LOW UNMATCHED TIME TRANS.
        PRINT DOUBLESPACED        /* PRINT TRANS. RECORD & DROP.
        GOTO 20.                  /* GO READ DET RECORD.
*
*    MATCHING RECORD PROCESSING FOLLOWS
*
    MOVE INF1-110    TO OFA1         /* MOVE ORIGINAL INPUT TO OUTPUT
    ADD  DET31-36-P  TO OFA31-36-P   /* UPDATE AMT FIELD
    WRITE OFA                        /* WRITE RECORD.
    GOTO 20                          /* GO READ DET RECORD.
9999END
```

Example 24

Print Report with OMIT, SORT AREA, SRTADJ and RPTSPCE

In this example, two simple payroll reports are easily produced using the REPORT statement, SRTADJ, and RPTSPCE. The input file is read and a report is produced. A second report is produced after the input file is sorted by employee name; a truncated record is passed to the SORT by the SORT AREA declarative. Note that the second PRINT REPORT statement omits certain fields (which have been truncated from the record).

VISION:Report Statements

```
OPTION SRTADJ=YES     /* OFFSET RELATIVE TO START OF AN AREA
OPTION RPTSPCE=4      /* SPACES IN BETWEEN FIELDS
OPTION BWZ=YES
OPTION LISTOPT=YES
    *-----------------------------------------------------*
    *                                                     *
    * SAMP24:   PRINT REPORT WITH OMIT, SORT AREA,        *
    *           SRTADJ AND RPTSPCE OPTIONS.               *
    *                                                     *
    *-----------------------------------------------------*
*INFDISC32000080SSYS010             /* IF VSE, REMOVE * AT POSITION 1
EQU PLANT           WST1-2
EQU DEPT            WST3-5
EQU EMP-NR          WST6-9
EQU EMP-NAME        WST10-25
EQU HR-RATE         WST46-49      2C
EQU HR-WORKED       WST50-53      2C
EQU GROSS-PAY       WST54-58-P    2C
*
SORT AREA (F) RL25       ON EMP-NAME PLANT DEPT
*
    HDR 1A 1
    HDR 2A 0EXAMPLE 24                  COMPUTER ASSOCIATES
*
*  SET UP REPORT HEADERS
REPORT DEPT (DEPT-NAME)             /* FIELD 1
       SPACE1                       /* SPACE BETWEEN FIELDS.
       EMP-NR (EMPLOYEE-NUMBER)     /* FIELD 2
       EMP-NAME (EMPLOYEE-NAME)     /* FIELD 3
       HR-WORKED (HOURS-WORKED)     /* FIELD 4
       GROSS-PAY                    /* FIELD 5
010  GET INF ATEND 400              /* GET DETAIL RECORD.
     MOVE INF1-80 TO WST1
     PRINT REPORT                   /* PRINT DETAIL (AND TOTALS).
     RELEASE WST1 TO SORT           /* PASS RECORD TO SORT
     GOTO 010                       /* GET ANOTHER RECORD.
400
     MOVE C'END OF FIRST REPORT' TO PRT1
     PRINT
     MOVE C'REPORT AFTER SORT'   TO PRT1
     PRINT
     DOHEADERS                      /* FORCE NEW HEADINGS NOW
420  RETURN SORTED INTO WST1        /* RECORD RETURNED FROM SORT
     IF @VAL-SORT-EOF  EQ C'E'      /* SORT EOF?
        GOTO EOJ.
```

```
    PRINT REPORT OMIT EMP-NR       /* DROP FIELDS NOT NEEDED
      HR-WORKED
      GROSS-PAY
    GOTO 420
99999999END
```

```
10/24/00                                              PAGE    1
EXAMPLE 24                 COMPUTER ASSOCIATES
DEPT EMPLOYEE        EMPLOYEE        HOURS            GROSS
NAME  NUMBER     ......NAME......   WORKED     .........PAY
005    0002      CLEARY,TOM          40.00        23,711.60
005    0008      RUNNINGTREE,TOM     41.00        32,523.04
008    0387      CLEVELAND,GROVER    46.00        40,496.72
008    0399      COCER,ONIES         44.00        26,972.88
008    0410      EVERS,HANK          40.00        35,933.92
010    0423      FAIR,MAXINE         42.00        49,218.40
010    0594      LADD,IDA            40.00        32,657.68
015    0620      LAFARY,ALFRED       42.00        44,535.92
015    0621      LANDERS,CAROL       36.00        47,453.12
015    0763      LANDERS,MICHAEL     42.00        41,394.32
015    0867      LAROCHELLE,RISA     44.00        51,791.52
120    1034      LAWSON,MOLER        54.00        60,662.80
END OF FIRST REPORT
REPORT AFTER SORT
10/24/00                                              PAGE    2
EXAMPLE 24                 COMPUTER ASSOCIATES
DEPT EMPLOYEE        EMPLOYEE        HOURS            GROSS
NAME  NUMBER     ......NAME......   WORKED     .........PAY
005              ATWATER,SCOTT
005              CLEARY,TOM
005              CLEGHORN,DELLA
008              CLEMENS,GARY
008              CLEVELAND,GROVER
008              COCER,ONIES
010              FAIR,MAXINE
005              JONES,LYLA
005              RUNNINGTREE,TOM
005              WINTERGARTEN,L.R
005              ZONK,HIERONYMOUS
```

The report output was truncated for conciseness.

Example 25

Print Report Summary

This example produces a report similar to the first report in Example 24, with one major difference. Here, the detailed report is not printed. PRINT REPORT SUMMARY allows you to obtain a summarized report of totals by changing only one line of an existing REPORT.

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP25:    "PRINT REPORT SUMMARY" TO ALLOW USERS TO      *
*            OBTAIN A SUMMARIZED REPORT OF TOTALS.         *
*                                                          *
************************************************************
*INFDISC32000080SSYS001            /* IF VSE, REMOVE * AT POSITION 1
EQU PLANT     INF1-2
EQU DEPT      INF3-5
EQU EMP-NAME  INF10-25
EQU HR-RATE   INF46-49    2C
EQU HR-WORKED INF50-53    2C
EQU GROSS-PAY INF54-58-P  2C
EQU AVG-RATE  CTC6-8-P    2C
EQU NR-EMP    CTD6-8-P
EQU SAVE-DEPT WST1-3      SPACES
*
SORT FILE INF ON PLANT DEPT
*
TITLE1 'COMPUTER ASSOCIATES'
TITLE2 'PAYROLL DISTRIBUTION FOR PLANT $PLANT$'
*
* SET UP REPORT HEADERS
*
REPORT DEPT (DEPT-NAME)               /* FIELD 1
       SPACE9                         /*   FORCE SPACING
       SAVE-DEPT (DEPT-NO)            /*
       SPACE5                         /*   FORCE SPACING
       HR-WORKED (HOURS-WORKED)       /* FIELD 2
       SPACE3                         /*   FORCE SPACING
       GROSS-PAY                      /* FIELD 3
       SPACE4                         /*   FORCE SPACING
       AVG-RATE (AVERAGE-HOURLY.RATE) /* FIELD 4
       SPACE5                         /*   FORCE SPACING
       NR-EMP (NUMBER-OF-EMPLOYEES)   /* FIELD 5
*
BREAK 1 DEPT SB 1 SA 1 PRINT C'DEPT TOTAL'
BREAK 2 PLANT SB 1 SA E PRINT C'PLANT TOTAL'
*
* TRACE ALL
010 GET INF ATEND EOJ                   /* GET DETAIL RECORD.
    CHECKBREAKS ON BREAKS PERFORM 100 THRU 200  /* AT BREAK
*                                               /*   PERFORM & PRINT
    MOVE  DEPT TO SAVE-DEPT           /* ** ADDED-SAVE FOR TOTALS TIME
    ACCUM HR-WORKED IN A 4 BYTE CTA   /* SET UP TOTALS FOR HR-WORKED
    ACCUM GROSS-PAY IN A 5 BYTE CTB   /* SET UP TOTALS FOR GROSS-PAY
    ACCUM NONE      IN A 3 BYTE CTC   /* SET UP TOTALS FOR AVG-RATE
    ACCUM ONE       IN A 3 BYTE CTD   /* SET UP NUM. OF EMPLOYEES
```

```
    PRINT REPORT SUMMARY              /* MOVE DEPT TO PRT AREA
    GOTO 010                          /* GET ANOTHER RECORD
*
*     COMPUTE WEIGHTED AVERAGE OF HOURLY RATE
*
100  DIVD CTB4-8-P 2D BY CTA5-8-P  0D GIVING CTC6-8-P 2D
     IF VAL180 EQ C'F'                /* ARE WE AT GRAND TOTALS?
        MOVE C'DIVISION TOTALS' TO PRT1.
200  EXIT
     GOTO EOJ
9999END
```

```
09/17/2002                COMPUTER ASSOCIATES                PAGE    1
                     PAYROLL DISTRIBUTION FOR PLANT 12
                                                                 NUMBER
DEPT             HOURS             GROSS          AVERAGE          OF
NAME           .WORKED      .........PAY      HOURLY.RATE      EMPLOYEES
DEPT TOTAL       54805        448,201.60             8.17             13
DEPT TOTAL       16200        145,471.04             8.97              4
DEPT TOTAL       16700        148,822.08             8.91              4
DEPT TOTAL       16400        185,174.88            11.29              4
DEPT TOTAL        5400         60,662.80            11.23              1
PLANT TOTAL     109505        988,332.40             9.02             26

09/17/2002                COMPUTER ASSOCIATES                PAGE    2
                     PAYROLL DISTRIBUTION FOR PLANT
                                                                 NUMBER
DEPT             HOURS             GROSS          AVERAGE          OF
NAME           .WORKED      .........PAY      HOURLY.RATE      EMPLOYEES
DIVISION TOTALS 109505        988,332.40             9.02             26
```

Some of the report output has been truncated for conciseness.

Example 26

SET PCC, MOVE VARIABLE LENGTH, EQU with Literals, Negative Numbers, WHEN and WHEN/REVERSE, QUIKVSAM with Read-Upd and Update

The following example updates an accounts receivable VSAM file (for example, CHARGE CARD). Each transaction is printed and SET PCC is used to suppress normal printer spacing, allowing payments to be underlined. Customer name format is reversed from last name first to first name last with the variable length MOVE and the REVERSE WHEN scan in lines 100 through 200, and then printed in the new format.

VISION:Report Statements

```
OPTION STMTEND=80,SEQCHK=NO
************************************************************
*                                                          *
* SAMP26:    UPDATE CHARGE MASTER VSAM FILE.  PRINT ALL    *
*            TRANSACTIONS.  USE SET PCC TO UNDER-SCORE     *
*            PAYMENTS ON A PRINT LINE.                     *
*                                                          *
*            ALSO HAS EXAMPLES OF VARIABLE LENGTH MOVE,    *
*            AND 'REVERSE WHEN' IN SCANNING CUSTOMER NAME, *
*            REVERSING FROM LAST NAME FIRST TO             *
*            FIRST NAME FIRST.                              *
*                                                          *
************************************************************
*INFDISC72800080SSYS012            /* If VSE, remove * at position 1
*
EQU AMOUNT  WST4-10 2C ZEROES
                                   /* Master amount total (PRT spec. 2c).
EQU NAME    WST20-39 SPACES                    /* Master name.
EQU CARD-NR WST11-19    S          /* Master card number (PRT spec. s).
*
EQU NAME-DET INF1-20               /* Update name.
EQU CARD-NR-DET (9)                /* Update card number.
EQU CODE-DET    (1)                /* Update transaction code.
EQU AMOUNT-DET  (5)-P 2C           /* Update transaction amount.
*
EQU LAST-LEN  SAV1-4-B  ZERO       /* Length of last name.
EQU FIRST-LEN (4)-B     ZERO       /* Length of first name.
EQU WHEN-LEN  VAL225-228-B         /* Length of when scan.
```

```
SORT FILE INF ON CARD-NR-DET        /* Sort update records on card number.
*
   HDR 1A 1 NAME              CARD-NUMBER  TRAN.      AMOUNT    BALANCE
     CALL QUIKVSAM C'PAYMFLE ' C'OPTION' SAV21
010  GET INF ATEND EOJ              /* Get detail record.
     MOVE NAME-DET TO PRT1          /* Print name as appears.
     WHEN NAME-DET INCLUDES C','    /* Scan for end of last name.
       SET PTA PTR2                 /* Save ptr of first name.
       MOVE WHEN-LEN TO LAST-LEN    /* Save length of last name.
       PERFORM 100 THRU 200         /* Move name to print.
       GOTO 020.                    /* Go get master record.
     MOVE NAME-DET TO PRT6          /* Last name only to print.
020  CALL QUIKVSAM C'PAYMFLE ' C'READ-UPD' WST1-80 CARD-NR-DET
     IF WST1-10 EQUAL HIVALUES      /* Find the master record?
     GOTO EOJ.                      /*    No, GOTO EOJ.
     MOVE CARD-NR TO PRT22          /* Move card number to print.
     MOVE AMOUNT-DET TO PRT42       /* Move transaction amount
     IF CODE-DET = C'P'             /* Is transaction a payment?
        MOVE C'PAYMENT' TO PRT35    /*    Yes, print c'payment'.
        MULT AMOUNT-DET 2D BY C'-1' 0D GIVING AMOUNT-DET 2D
         GOTO 030.                  /* Make amount negative.
     MOVE  C'CHARGE' TO  PRT35      /* No, print c'charge'.
030  ADD AMOUNT-DET TO AMOUNT       /* Add transaction amount to total
     MOVE AMOUNT TO PRT58           /* Move total amount due to print
     PRINT DOUBLESPACED             /* Print detail line double spaced
     IF CODE-DET EQ C'P'            /* Is transaction a payment?
        SET PCC SINGLESPACED        /* Yes, set carriage control to
        MOVE C'-' TO PRT1           /* Suppress spacing in order to
        MOVE PRT1-65 TO PRT2        /* Underscore payment transactions
        PRINT                       /* Print underscore, with no spacing
        SET PCC OFF.                /* Turn off user carriage control.
     CALL QUIKVSAM C'PAYMFLE ' C'UPDATE' WST1-80 /*
     GOTO 010                       /* Update master record, go get DET
*
100  WHEN NAME-DET INCLUDES NONSPACES REVERSE
                                    /* Scan for end of name
       MOVE C'20' TO FIRST-LEN      /* Calc length of first name.
       SUB LAST-LEN FROM FIRST-LEN  /* Sub length of last name.
       SUB WHEN-LEN FROM FIRST-LEN  /* Sub length of space at eor.
       MOVE PTA1 TO PRT69 FIRST-LEN. /* Move first name to PRT.
     WHEN PRT69-89 INCLUDES NONBLANKS REVERSE    /* Scan for end of name
        MOVE NAME-DET TO PRT1 LAST-LEN.         /* Move last name to PRT
200 EXIT
    GOTO EOJ
9999END
```

Example 27

Native VSAM Using GET, SET PTA, PRINTHEX

The following example reads a VSAM file sequentially using the GET verb, and prints the variable length record using the PRINTHEX statement.

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP27:    READ VSAM FILE (KSDS) SEQUENTIALLY, USING     *
*            NATIVE VSAM.                                  *
*                                                          *
************************************************************
*INFKSDS    0080                  /* If VSE, remove * at position 1
    OPEN INF
    SET PTA INF1
    SET PTA DOWN 2                /* PTA now points to length field.
040 GET
    PRINTHEX INF1 PTA1-2-B
    GO TO 040
9999END
```

Example 28

Native VSAM (RRDS) Using GET and SETGENKEY

The following example starts sequential retrieval of an RRDS VSAM file at record 1000, using the SETGENKEY verb follow by the GET verb.

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP28:    READ VSAM FILE (RRDS), STARTING WITH RECORD   *
*            1000.  USE THE SETGENKEY VERB TO POSITION     *
*            VSAM FILE, THEN FOLLOW WITH GET TO READ       *
*            RECORDS SEQUENTIALLY THEREAFTER.              *
*                                                          *
************************************************************
*INFRRDS    0080                  /* If VSE, remove * at position 1
EQU RRDS-KEY WST1-4-B
TITLE 'RRDS FILE LISTING'
REPORT INF1-80 (RRDS.RECORD.IMAGE)
    MOVE P'10' TO RRDS-KEY        /* Position to 10th record.
    SETGENKEY INF USING RRDS-KEY  /* Equal is the default.
030 GET
    PRINT REPORT
    GO TO 030
9999END
```

Example 29

Native VSAM (RRDS) Using WRITE

The following example loads an RRDS VSAM data set with a record size of 20 bytes. The field, PRODUCT-NUMBER, is created by the program at the time of the load. All records are printed as they are loaded.

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP29:    WRITE AN RRDS VSAM FILE USING NATIVE VSAM     *
*                                                          *
************************************************************
*INFCARD                          /* IF VSE, REMOVE * AT POSITION 1
*OFARRDS    0020                  /* IF VSE, REMOVE * AT POSITION 1
*** DEFINE INPUT FILE
EQU INPUT-RECORD   INF1-20
*** DEFINE RRDS PRODUCT FILE
EQU PRODUCT-RECORD OFA1-20
EQU PRODUCT-NAME   (PRODUCT-RECORD)     /* REDEFINE.
***
EQU WORKINGSTORAGE WST
EQU PRODUCT-NUMBER (4)-B     ZEROES
***
    TITLE 'PRODUCT FILE LOAD'
    REPORT PRODUCT-NUMBER
           PRODUCT-NAME
*** SET UP RECORD LENGTH FOR OFA AS 20 BYTES LONG

    OPEN OFA
    SET PTA OFA1
    SET PTA DOWN 2
    MOVE P'20' TO PTA1-2-B
*** READ THE INPUT CARDS AND LOAD THE RRDS FILE
100 GET
    MOVE INPUT-RECORD TO PRODUCT-RECORD
    ADD C'1' TO PRODUCT-NUMBER
    WRITE OFA
    PRINT REPORT
    GO TO 100
9999END
WIDGETS
ROLLER SKATES
SKATEBOARDS
KITES
SURF BOARDS
/*
```

Example 30

Native VSAM (KSDS, RRDS, ESDS) Using Random Access, READ, ADDRECORD, REWRITE, DELETE

This example shows an update of a VSAM KSDS file. Records are added, deleted, or updated to the customer master file, or an invoice or payment record is created, all based upon the INF action-code. A report is printed, showing all transactions processed.

VISION:Report Statements

```
************************************************************
*                                                          *
* SAMP30:    UPDATE CUSTOMER MASTER (DET KSDS),            *
*            ADD A NEW RECORD TO CUSTOMER MASTER FILE (DET)*
*            DELETE A CUSTOMER MASTER RECORD, OR           *
*            CREATE AN INVOICE RECORD (OFA ESDS),  OR      *
*            CREATE A PAYMENT RECORD  (OFA ESDS).          *
*                                                          *
*            ACTION ON WHAT TO DO IS BASED UPON INF RECORD.*
*                                                          *
************************************************************
*INFCARD                          /* IF VSE, REMOVE * AT POSITION 1
*DETVSAM                          /* IF VSE, REMOVE * AT POSITION 1
*INCVSAM                          /* IF VSE, REMOVE * AT POSITION 1
*OFAVSAM                          /* IF VSE, REMOVE * AT POSITION 1
EQU ACTION-RECORD  INF          /* DEFINE INPUT ACTION RECORD.
EQU ACTION-CODE    (1)
EQU ACTION-NUMBER  (5)
EQU ACTION-PRODUCT (2)
EQU ACTION-AMOUNT  (5)
EQU ACTION-NAME    (30)

EQU MASTER-RECORD  DET          /* DEFINE CUSTOMER MASTER RECORD.
EQU MASTER-NUMBER  (5)
EQU MASTER-NAME    (30)
EQU MASTER-BALANCE (6)
EQU FILLER         (MASTER-RECORD)
EQU MASTER-SPACES  (41)

EQU PRODUCT-RECORD INC          /* DEFINE PRODUCT DESCRIPTION RECORD.
EQU PRODUCT-NUM    (2)
EQU PRODUCT-NAME   (18)

EQU INVCPAY-RECORD OFA          /* DEFINE INVOICE AND PAYMENT RECORD.
EQU INVCPAY-TYPE   (1)
EQU INVCPAY-NUMBER (5)
EQU INVCPAYPRODUCT (2)
EQU INVCPAY-AMOUNT (5)
EQU INVCPAY-NAME   (30)

EQU WORKINGSTORAGE WST          /* DEFINE WORKING STORAGE FIELDS.
EQU MESSAGE        (20)
EQU PRODUCT-NUMBER (4)
***    DEFINE TITLE AND REPORT
001 TITLE 'CUSTOMER MASTER UPDATE LISTING'
002 REPORT ACTION-CODE ACTION-NUMBER ACTION-AMOUNT ACTION-PRODUCT
```

```
003         PRODUCT-NAME MASTER-NAME MASTER-BALANCE MESSAGE
***    OPEN FILES AND SET RECORD LENGTHS
***    OPEN DET C'PASSWORD'

    OPEN DET
    SET PTA DET1
    SET PTA DOWN 2 /* POINT TO DET'S RECORD LENGTH FIELD.
    OPEN OFA
    SET PTB OFA1
    SET PTB DOWN 2
    MOVE C'43' TO PTB1-2-B   /* INITIALIZE OFA'S RECORD LENGTH FIELD.
***    PROCESS INPUT ACTION REQUESTS
100 GET
    IF ACTION-CODE EQ C'A'           /* A = ADD A NEW CUSTOMER MASTER.
       PERFORM 200 THRU 299
       GO TO 175.
    IF ACTION-CODE EQ C'C'           /* C = CHANGE A CUSTOMER MASTER.
       PERFORM 300 THRU 399
       GO TO 175.
    IF ACTION-CODE EQ C'D'           /* D = DELETE A CUSTOMER MASTER.
       PERFORM 400 THRU 499
       GO TO 175.
    IF ACTION-CODE EQ C'I' OR        /* I = CREATE INVOICE RECORD.
       IF ACTION-CODE EQ C'P'        /* P = CREATE PAYMENT RECORD.
           PERFORM 500 THRU 699
           GO TO 175.
    MOVE C'INVALID ACTION CODE' TO MESSAGE
175 PERFORM 700 THRU 799             /* CREATE A REPORT LINE.
    GO TO 100
***   ADD A NEW CUSTOMER MASTER RECORD
200 MOVE ACTION-NUMBER TO MASTER-NUMBER
    MOVE ACTION-NAME TO MASTER-NAME
    MOVE ZEROES TO MASTER-BALANCE
    MOVE C'41' TO PTA1-2-B
    ADDRECORD DET ONERROR 260
    MOVE C'MASTER ADDED' TO MESSAGE
    GO TO 299
260 IF @VAL-VSAM-ERR EQ C'DUP'
    MOVE C'DUPLICATE RECORD' TO MESSAGE.
299 EXIT
***     CHANGE AN EXISTING CUSTOMER MASTER RECORD
300 READ DET USING ACTION-NUMBER ONERROR 350
    MOVE ACTION-NAME TO MASTER-NAME
    REWRITE DET
    MOVE C'NAME CHANGED' TO MESSAGE
    GO TO 399
350 IF @VAL-VSAM-ERR EQ C'RNF'
        MOVE C'RECORD NOT FOUND 1' TO MESSAGE.
399 EXIT
***     DELETE AN EXISTING CUSTOMER MASTER RECORD
400 READ DET USING ACTION-NUMBER ONERROR 440
    DELETE DET
    MOVE C'RECORD DELETED' TO MESSAGE
    GO TO 499
440 IF @VAL-VSAM-ERR NOT EQ C'OK '
        MOVE C'RECORD NOT FOUND 2' TO MESSAGE.
499 EXIT
***     CREATE AN INVOICE OR PAYMENT RECORD
500 READ DET USING ACTION-NUMBER ONERROR 600
    IF ACTION-AMOUNT NOT NUMERIC
        MOVE C'AMOUNT NOT NUMERIC' TO MESSAGE
        GO TO 699.
    IF ACTION-PRODUCT NOT NUMERIC
        MOVE C'PRODUCT NOT NUMERIC' TO MESSAGE
        GO TO 699.
```

```
    MOVE ACTION-CODE TO INVCPAY-TYPE
    MOVE ACTION-NUMBER TO INVCPAY-NUMBER
    MOVE MASTER-NAME TO INVCPAY-NAME
    MOVE ACTION-AMOUNT TO INVCPAY-AMOUNT
    IF ACTION-CODE EQ C'I'
        MOVE C'INVOICE NOTED' TO MESSAGE
        ADD ACTION-AMOUNT TO MASTER-BALANCE.
    IF ACTION-CODE EQ C'P'
        MOVE C'PAYMENT NOTED' TO MESSAGE
        SUB ACTION-AMOUNT FR MASTER-BALANCE.
    WRITE INVCPAY-RECORD                    /* OFA FILE
    REWRITE DET
    GO TO 699
600 IF @VAL-VSAM-ERR NOT EQ C'OK '
        MOVE C'RECORD NOT FOUND 3' TO MESSAGE.
699 EXIT
***     CREATE A REPORT LINE
700 IF @VAL-VSAM-ERR EQ C'RNF'
       MOVE SPACES TO MASTER-SPACES.
    IF ACTION-PRODUCT NOT NUMERIC
       GO TO 760.
    IF ACTION-PRODUCT ZERO
       GO TO 760.
    MOVE ACTION-PRODUCT TO PRODUCT-NUMBER
    MOVE SPACES TO PRODUCT-NAME
    READ INC USING PRODUCT-NUMBER ONERROR 740
    GO TO 770
740 IF @VAL-VSAM-ERR EQ C'RNF'
        MOVE @VAL-VSAM-ERR TO PRT1
        PRINT
        MOVE C'RECORD NOT FOUND 4' TO MESSAGE.
    GO TO 770
760 MOVE C'*INVALID PRODUCT*' TO PRODUCT-NAME
770 PRINT REPORT DOUBLESPACED
    MOVE SPACES TO MASTER-SPACES
    MOVE SPACES TO PRODUCT-NAME
    MOVE SPACES TO MESSAGE
799 EXIT
9999END    /* FOLLOWING INPUT IS THE ACNT  */
A00015  01500ALWAYS OPEN CAFE               INVALID PRODUCT
A00018  00200OLD ENGLISH PIPE SHOPPE        NEW RECORD
C00010       EARLY MORNING BAKERY           INVALID PRODUCT
C0001001     SUNDAY MORNING BAKERY
D00051       VERY BEST PIE SHOP             INVALID PRODUCT
I00020BB10000FARMERS MARKET PRODUCE         INVALID PRODUCT
I0002088810000FARMERS MARKET PRODUCE
I0002001l0000FARMERS MARKET PRODUCE
P00040  0S000GREAT NORTHWEST CANNERY        INVALID PRODUCT
P00040  05000GREAT NORTHWEST CANNERY        INVALID PRODUCT
P000405000999GREAT NORTHWEST CANNERY        PAYMENT
/*
```

Example 31

Native VSAM using GET, QUIKIPDS, WHEN with INCLUDES/OMITS, WHEN/REVERSE, IF...NUMERIC, IF...ALPHA, Negative IF, Multiple HDR with $names$

This example reads the AR File, using native VSAM. Several examples of IF statements are shown, including IF ... NUMERIC, IF ... ALPHA, as well as negative IF statements such as IF ... NOT NUMERIC and IF ... NOT ALPHA. Multiple HDR statements (HDR1A-HDR3B) are shown, with various $names$, such as $IPLDAT$, TIM, $JOBNAM$, and PG. These $names$ are reserved words, and VISION:Report fills the area specified by these reserved words with items such as the IPL date, time.

The OPTION LIST=NO statement specifies that the VISION:Report program is not to be printed. This is normally done in a production environment, where the program listing is not required. However, in testing or debugging mode, the LIST=NO should be removed.

VISION:Report Statements

```
OPTION LIST=NO
OPTION UEXIT1=QUIKIPDS   /* UEXIT1=QUIKIPDS IF MVS; VSE IS QUIKINCL
*********************************************************************
*                                                                   *
* SAMP31:   READ ARFILE (VSAM).                                     *
*        VERBS:  SIMPLE GET AND PRINT I/O VERBS.                    *
*                                                                   *
*                SCAN CUST-NAME AND SWITCH AROUND FIRST/LAST NAMES  *
*                WITH 'WHEN' VERB.                                  *
*                                                                   *
*                NUMERIC/ALPHA CHECKS, NEGATIVE & POSITIVE          *
*                    IF .. ALPHA, IF.. NUMERIC, IF..NOT ...         *
*                                                                   *
*                MVS: USES UEXIT1=QUIKIPDS AND ++INCLUDE ARDEFINE   *
*                VSE: USES UEXIT1=QUIKINCL AND ++INCLUDE Q.ARDEFINE*
*                     USES OPTION OF LIST=NO,                       *
*                AND ON 'HDR', $IPLDAT$, $TIM$, $JOBNAM$, AND $PG$.*
*                                                                   *
*********************************************************************
*INFKSDS    0352                  /* IF VSE, REMOVE * AT POSITION 1
EQU  AR-ENT-REC         INF
++INCLUDE ARDEFINE     /* ++INCLUDE Q.ARDEFINE   IF VSE

EQU  LENGTHS            WST101-108        /* ENTIRE LENGTHS
EQU  LAST-LEN           WST101-104-B      /* LEN FOR LAST NAME
EQU  FIRST-LEN          WST105-108-B      /* LEN FOR FIRST NAME

EQU  FIRST-NAME         PRT14-24
EQU  LAST-NAME          PRT26-40

EQU  WHEN-LEN           VAL225-228-B      /* LEN AFTER USING WHEN

    HDR 1A 1  IPLDATE= $IPLDAT$    TIME= $TIM$   JOB NAME= $JOBNAM$
    HDR 1B  SAMPLE31               PAGE= $PG$
    HDR 2A 0SAMPLE PROGRAM USES "GET, PRINT, IF NUMERIC, IF ALPHA,
    HDR 2B  IF ...NEGATIVE, WHEN IN SWITCHING LAST/FIRST NAME
```

```
    HDR 3A 0CODE ACCOUNT FIRST NAME  LAST NAME        STREET
    HDR 3B    CITY                ST  ZIP

010 GET INF  ATEND  EOJ
    MOVE ZEROS   TO LAST-LEN             /* CLEAR OUT LEN FIELDS
    MOVE ZEROS   TO FIRST-LEN
    MOVE ZEROS   TO WHEN-LEN
    MOVE SPACES  TO FIRST-NAME
    MOVE SPACES  TO LAST-NAME
    IF  AR-CUST-NAME EQ SPACES           /* SKIP ANY WITH THIS
        MOVE C'*****'   TO FIRST-NAME
        MOVE C'*****'   TO LAST-NAME
        GOTO 40.                         /* SKIP "WHEN" VERB

* SCAN AR-CUST-NAME, LOOKING FOR A COMMA, PERIOD, OR A BLANK
* THIS THEN SEPARATES LAST NAME FROM FIRST NAME
* N-O-T-E:  IN TRYING TO DOCUMENT EACH LINE OF CODE,
*           IT BECAME NECESSARY TO INTERSPERSE COMMENTS
*           IN BETWEEN LINES OF CODE
*
*           BE CAREFUL OF WHEN THE "WHEN" VERB ACTUALLY ENDS !!
*
* NOTE:  WHEN YOU'VE GOT A NAME SUCH AS "S.F.MEM.HOSP"
*        OR "SANTA FE MEMORIAL HOSPITAL",
*        IT WILL NOT COME OUT PROPERLY,     <==========
*        BASED UPON THE CURRENT CODING!

    WHEN  AR-CUST-NAME INCLUDES  C',' OR /* SCAN FOR COMMA
    WHEN  AR-CUST-NAME INCLUDES  C'.' OR /* SCAN FOR PERIOD
    WHEN  AR-CUST-NAME INCLUDES  C' '    /* TRY FOR BLANK THEN
        SET PTA     PTR2                 /* SAVE PTR TO SEPARATOR
        MOVE  WHEN-LEN   TO LAST-LEN     /* SAVE LAST NAME LENGTH

* NEXT CODE SETS UP MAX LENGTH FOR FIRST NAME
* IT SHOULD BE 1 LESS THAN THE ENTIRE LENGTH OF FIELD

        MOVE  C'24'      TO FIRST-LEN    /* SET MAX FOR FIRST NAME

***  SCAN END OF NAME NEXT  (CONTINUING OF FIRST 'WHEN' VERB)

        WHEN  AR-CUST-NAME INCLUDES NONBLANK REVERSE
            SUB WHEN-LEN      FROM FIRST-LEN
            SUB LAST-LEN      FROM FIRST-LEN
            MOVE PTA1           TO FIRST-NAME   FIRST-LEN
            MOVE AR-CUST-NAME   TO LAST-NAME    LAST-LEN.

* THE FOLLOWING DOES SOME ALPHA/NUMERIC/SPACES  CHECK.

040  IF AR-ACCT-CODE   IS NOT ALPHA     /* TRY NEGATIVE 'NOT ALPHA'
         MOVE  C'*****'     TO   PRT2
         GOTO 50.
     MOVE  AR-ACCT-CODE     TO   PRT2
050  IF AR-ACCOUNT     IS NOT NUMERIC   /* NEGATIVE NUMERIC CHECK
         MOVE  C'*****'     TO   PRT6
         GOTO 60.
     MOVE  AR-ACCOUNT       TO   PRT6
060  IF AR-STREET      IS NOT = SPACES  /* STREET SHOULDN'T BE BLANKS
         MOVE  AR-STREET    TO   PRT43
         GOTO 070.
     MOVE  C'*****'         TO   PRT43
070  IF AR-CITY        IS NOT = SPACES  /* SHOULDN'T BE BLANKS
         MOVE  AR-CITY      TO   PRT70
         GOTO 080.
     MOVE  C'*****'         TO   PRT70
080  IF AR-STATE       IS ALPHA         /* SHOULD BE ALPHA
```

```
          MOVE  AR-STATE     TO   PRT90
          GOTO 090.
     MOVE C'**'              TO   PRT90  /* STATE SHOULD BE ALPHA
090  IF AR-ZIP          IS  NUMERIC      /* CHECK IF NUMERIC
          MOVE AR-ZIP        TO   PRT94
          GOTO 100.
     MOVE C'*****'           TO   PRT94  /* ZIP CODE NOT NUMERIC
100  PRINT
     GOTO 010

9999END
```

```
IPLDATE= 10/25/00    TIME= 14:33   JOB NAME= QJSMP31            SAMPLE31              PAGE=    1
 SAMPLE PROGRAM USES "GET, PRINT, IF NUMERIC, IF ALPHA, IF NEGATIVE
 CODE ACCOUNT FIRST NAME   LAST NAME       STREET                    CITY               ST  ZIP
  BO  8006547 ERNESTO      TORRES          23444 PARK LANE           LOS ANGELES        CA  *****
  EO  6002587 FE HOSP ASSN SANTA           1212 WISCONSIN DRIVE      LOS ANGELES        CA  80023
  EO  6208657 SUH          CHO PYUNG       33333 PALL MALL           GLENDALE           CA  91206
  EO  7082509 F.MEM.HOSP.  S               900 JOHN WAYNE STREET     NILES              IL  49121
  FO  6024963 GARY E       HILL            5445 COVENTRY ST          TEMPLE CITY        CA  91780
  FO  6044395 MICHEAL S    CANO            56479 BOND STREET #12     BREA               CA  92621
  FO  6059708 RAY          CHAVEZ          7986 MAYFLOWER AVE        MONTEREY PARK      CA  *****
  FO  6095631 MICHAEL      TODIPE          5678 REGENT SQR           INGLEWOOD          CA  *****
  FO  6107265 G J          GENVARDI        1220 RIVERWAY STREET      INDIANAPOLIS       IN  46220
  FO  6123228 JULIAN       SILVA           33356 WIZZY DR #32        LOS ANGELES        CA  90033
  FO  8011508 RAY          HUGHES          3025 WINTER ST            MONTEREY PARK      CA  91754
  IO  2002299 ORTEGA       PLACIDO         1247 S ST LOUIS ST        LOS ANGELES        CA  90023
  IO  6009166 JEFFREY      LOCKE           321 PALMOROY AVE          ROSEMEAD           CA  *****
  IO  6112536 NORMA A      CHAVEZ          2483 BUCKTOWN             HACIENDA HTS       CA  91745
  IO  6123317 IRENE        VASGUEZ         329 W 7TH ST              CITY OF COMMERCE   CA  90040
  NA  8003173 *****        *****           1008 84TH                 MONTEREY PARK      CA  91754
  NA  9010033 *****        *****           3735 S VICTORIA AVE       LOS ANGELES        CA  90023
  PO  1001191  MARY        CARDOZA         1008 WESTMORELAND         LA                 CA  90007
  PO  1006681  EARL        MYERS           3735 84TH                 LA CANADA          CA  91010
  PO  6023185 JOSEPHINE G  PORTWOOD        18892 ORCHARD AVE         LOS ANGELES        CA  *****
  PO  6099963 JESUS        ROMO            2432 S VICTORIA AVE       ORANGE CITY        CA  *****
  PO  6216846 PAUBLINA     TORRES          601 E AVE 28              LOS ANGELES        CA  90063
  PO  7030142  JEWEL       HARRIS          331 1/2 EAST FIRST        LA                 CA  90031
  PO  7053185  MARY        SMITH           280 DOLPHIN CAUSEWAY      DOLPHIN            AL  36660
  PO  7064535 *****        *****           1946 HORBART              CITY OF COMMERCE   CA  90040
  PO  7068794  MERIAN      BACTAD          573 SO BOYLE AVE          CHINA LAKE         CA  93555
  PO  7070721 *****        *****           610 W 43RD PL             HACIENDA HTS       CA  91745
  WO  8011036 CHARLES      ANDREWS         1310 LOMA DR              CAMBRIA PINES      CA  *****
  WO  8011699 FRANK        OBRESON         345 CHICAGO               WOODRIDGE          IL  60517
  WO  8031096 ARMENDO      ROMERO          1730 TIMBERLAND AVENUE    TINLEY PARK        IL  60477
  WO  9007156 ROAD CO.     KANGAROO R      100 ALCOA PLAZA           BALTIMORE          MD  21227
  WO  9009469 MUT OF WAUSAUEMP             3970 18TH                 LA                 CA  90005
  WO  9020039 MUTUAL INS   LIBERTY         500 SOUTH UNIVERSITY ST   INDIANAPOLIS       IN  46202
```

Example 32

Native VSAM (ESDS) with Alternate Index, Using OPEN/CLOSE, GET, READ, SETGENKEY, REWRITE, SET PTA

This example uses a native VSAM (ESDS) with alternative index, along with other VSAM verbs. First, the DET file is read sequentially using GET, then read randomly using READ and SETGENKEY with GET. The record is updated by the REWRITE verb. The OFA file has one record added and finally the INF file is read sequentially. The various files are opened and closed at different times and many of the VSAM verbs use the ONERROR option. A print trail shows the flow of the program through its various stages. Note that files DET and OFA are the same.

Prior to running the VISION:Report, an IDCAMS job similar to the following example was run (although the IDCAMS is a MVS job stream, most, if not all the IDCAMS statements, would be identical under VSE).

```
DELETE   ESDS.TEST CLUSTER
SET MAXCC=0
(NAME(ESDS.TEST) -
                    VOLUMES(volser) -
                    RECORDSIZE(80 80) -
                    NONINDEXED) -
           DATA    (NAME(ESDS.TEST.DATA) -
                    CISZ(512) -
                    TRACKS(1 1))
REPRO    INFILE(SYSUT1) -
           OUTDATASET(ESDS.TEST)
PRINT    INDATASET(ESDS.TEST) -
           CHARACTER
DEFINE   AIX     (NAME(ESDS.TEST.AIX) -
                    RELATE(ESDS.TEST) -
                    VOLUMES(volser) -
                    KEYS(10 0) -
                    RECSZ(128 256) -
                    TRACKS(1 1) -
                    UPGRADE) -
           DATA    (NAME(ESDS.TEST.AIX.DATA) -
                    CISZ(1024)) -
           INDEX   (NAME(ESDS.TEST.AIX.INDEX) -
                    CISZ(512))
DEFINE   PATH    (NAME(ESDS.TEST.PATH) -
                    PATHENTRY(ESDS.TEST.AIX) -
                    UPDATE)
BLDINDEX INDATASET(ESDS.TEST) -
           OUTDATASET(ESDS.TEST.AIX) –
           INTERNALSORT
```

The input to build the ESDS file is as follows:

```
BBBB      TEST RECORD NUMBER 1
CCCC      TEST RECORD NUMBER 2
CCCCCCCCCCTEST RECORD NUMBER 3
BBBBBBBBBBTEST RECORD NUMBER 4
AAAAAAAAAATEST RECORD NUMBER 5
```

VSE JCL Example

```
// JOB SAMP32
// DLBL  filename,'your.VISION.lib'
// EXTENT    ,volser
// LIBDEF  PHASE,SEARCH=(lib.sublib)
// DLBL    INF,'ESDSAIX',,VSAM
// DLBL    OFA,'ESDSAIX.PATH',,VSAM
// DLBL    DET,'ESDSAIX.PATH',,VSAM
// DLBL    OFB,'ESDSAIX',,VSAM
* STEP1 - CREATE VSAM ESDS AND ALTERNATE INDEX
/*
// EXEC PGM=IDCAMS,SIZE=AUTO
.
.
.
  REPRO    INFILE(SYSIPT)                          -
           OUTFILE(OFB)
BBBB      TEST RECORD NUMBER 1
CCCC      TEST RECORD NUMBER 2
CCCCCCCCCCTEST RECORD NUMBER 3
BBBBBBBBBBTEST RECORD NUMBER 4
AAAAAAAAAATEST RECORD NUMBER 5
/*
// EXEC PGM=IDCAMS,SIZE=AUTO
  DEFINE   ...
.
.
.
/*
// EXEC QUKBJOB,SIZE=512K
... VISION:Report statements as shown below
/*
/&
```

MVS JCL Example

```
//SAMP32   JOB  (800-0000,0000),'Example 33'
//STEP1   EXEC  QJTEST
//QJ.SYSUT1   DD  DSNAME=ESDS.TEST,DISP=SHR
//QJ.SYSUT2   DD  DSNAME=ESDS.TEST.AIX,DISP=SHR
//QJ.SYSDET   DD  DSNAME=ESDS.TEST.AIX,DISP=SHR
//QJ.SYSIN    DD  *
  ...  VISION:Report statements as shown below
/*
//
```

VISION:Report Statements

```
OPTION TRACECT=1
*-------------------------------------------------------*
* SAMP32:     ESDS WITH ALTERNATE INDEX                 *
*             USING NATIVE VSAM.                        *
*                                                       *
*  VISION:REPORT VSAM ESDS ALTERNATE INDEX TEST         *
*  ---------------------------------------              *
*  GET                    CLOSE                         *
*  OPEN                   READ                          *
*  SETGENKEY              REWRITE                       *
*  WRITE                                                *
*-------------------------------------------------------*
*  OUTPUT FILES OFB, OFC, AND OFD ARE NOT USED.         *
```

```
*  INPUT FILES INC AND IND ARE NOT USED.              *
*                                                     *
*  FILE INF IS A NONEMPTY VSAM ESDS CLUSTER.          *
*                                                     *
*  FILE DET IS A PATH FOR AN ALTERNATE INDEX TO       *
*  THE INF CLUSTER.                                   *
*                                                     *
*  FILE OFA IS THE SAME PATH AS FILE DET.  <====      *
*-----------------------------------------------------*
*INFESDS    0040             /* IF VSE, REMOVE * AT POSITION 1
*OFAESDS    0040             /* IF VSE, REMOVE * AT POSITION 1
*DETESDS    0040             /* IF VSE, REMOVE * AT POSITION 1
     EQU WST-KEY     WST1-10
     EQU WST-KEY
     EQU WST-KEY-B   WST1-4-B   /* REDEFINE TO FOOL IT  */
     TITLE '$IPLDAT$               VSAM ESDS ALTERNATE INDEX     '
     TITLE '          PAGE $PG$'
    *--------------------------------------*
    *  VSAM ESDS GET USING ALTERNATE INDEX  *
    *--------------------------------------*
*    TRACE ALL
     MOVE C'VSAM ESDS GET' TO PRT1-13
     MOVE C'USING ALTERNATE INDEX' TO PRT15
     PRINT
     OPEN DET
1    GET DET ATEND 2
     IF VAL247-247-B NOT = ZERO
       GO TO 11.
     MOVE DET1-40 TO PRT1-40
     PRINT
     GO TO 1
2    MOVE C'END OF FILE DET' TO PRT1-15
     PRINT DOUBLE SPACED
    *-------------------------------------*
    *  CLOSE AND RE-OPEN FILE AFTER EOF  *
    *-------------------------------------*
     MOVE C'CLOSE DET' TO PRT1
     PRINT
     CLOSE DET
     MOVE C'OPEN DET' TO PRT1
     PRINT
     OPEN DET
    *---------------------------------------*
    *  VSAM ESDS READ USING ALTERNATE INDEX  *
    *---------------------------------------*
     MOVE C'VSAM ESDS READ' TO PRT1-14
     MOVE C'USING ALTERNATE INDEX' TO PRT16
     PRINT DOUBLE SPACED
     MOVE C'SEARCH KEY = CCCCCCCCCC' TO PRT1-23
     PRINT
     MOVE C'CCCCCCCCCC' TO WST-KEY
     READ DET USING WST-KEY-B EQUAL ONERROR 5
     MOVE DET1-40 TO PRT1-40
     PRINT DOUBLE SPACED
    *-----------------------------------------------*
    *  VSAM ESDS SETGENKEY USING ALTERNATE INDEX  *
    *-----------------------------------------------*
     MOVE C'VSAM ESDS SETGENKEY' TO PRT1-19
     MOVE C'USING ALTERNATE INDEX' TO PRT21
     PRINT DOUBLE SPACED
     MOVE C'SEARCH KEY = BBBBBBBBBB' TO PRT1-23
     PRINT
     MOVE C'BBBBBBBBBB' TO WST-KEY
     SETGENKEY DET USING WST-KEY-B EQUAL ONERROR 4
     GET DET
```

```
     IF VAL247-247-B NOT = ZERO
       GO TO 11.
     MOVE DET1-40 TO PRT1-40
     PRINT DOUBLE SPACED
     GET DET
     IF VAL247-247-B NOT = ZERO
       GO TO 11.
     MOVE DET1-40 TO PRT1-40
     PRINT
    *-----------------------------------------*
    *  VSAM ESDS REWRITE USING ALTERNATE INDEX  *
    *-----------------------------------------*
     MOVE C'VSAM ESDS REWRITE' TO PRT1-17
     MOVE C'USING ALTERNATE INDEX' TO PRT19
     PRINT DOUBLE SPACED
     MOVE SPACES TO DET31-40
     MOVE C' REWRITE' TO DET31
     MOVE DET1-40 TO PRT1
     PRINT DOUBLE SPACED
     REWRITE DET ONERROR 7
     MOVE C'REWRITE SUCCESSFUL' TO PRT1
     PRINT DOUBLE SPACED
     MOVE C'CLOSE DET' TO PRT1
     PRINT DOUBLE SPACED
     CLOSE DET
    *----------------------------------------*
    *  VSAM ESDS WRITE USING ALTERNATE INDEX  *
    *----------------------------------------*
     MOVE C'OPEN OFA' TO PRT1
     PRINT
     OPEN OFA
     MOVE C'VSAM ESDS WRITE' TO PRT1-15
     MOVE C'USING ALTERNATE INDEX' TO PRT17
     PRINT DOUBLE SPACED
     MOVE C'# 9' TO OFA1-10
     MOVE C'TEST RECORD NUMBER 6' TO OFA11-30
     MOVE C' WRITE    ' TO OFA31-40
     SET PTA OFA1
     SET PTA DOWN 2
     MOVE P'40' TO PTA1-2-B
     MOVE OFA1-40 TO PRT1-40
     PRINT DOUBLE SPACED
     WRITE OFA ONERROR 10
     MOVE C'WRITE SUCCESSFUL' TO PRT1
     PRINT DOUBLE SPACED
     MOVE C'CLOSE OFA' TO PRT1
     PRINT DOUBLE SPACED
     CLOSE OFA
    *-----------------------------*
    *  GET VSAM ESDS SEQUENTIALLY  *
    *-----------------------------*
     MOVE C'OPEN INF' TO PRT1
     PRINT
     OPEN INF
     MOVE C'GET VSAM ESDS SEQUENTIALLY' TO PRT1-26
     PRINT DOUBLE SPACED
     PRINT
3    GET INF ATEND 13
     IF VAL247-247-B NOT = ZERO
       GO TO 11.
     MOVE INF1-40 TO PRT1-40
     PRINT
     GO TO 3
    *----------------------*
    *  ONERROR PROCEDURES  *
```

```
    *----------------------*
4    MOVE C'SETGENKEY DET FAILED' TO PRT1
     GO TO 6
5    MOVE C'READ DET FAILED' TO PRT1
6    PRINT DOUBLE SPACED
     MOVE C'KEY =' TO PRT1-5
     MOVE WST-KEY TO PRT7
     PRINT
     GO TO 12
7    MOVE C'REWRITE DET FAILED' TO PRT1
     PRINT DOUBLE SPACED
     GO TO 12
10   MOVE C'WRITE OFA FAILED' TO PRT1
     PRINT DOUBLE SPACED
     GO TO 12
11   MOVE C'GET FAILED' TO PRT1
     PRINT DOUBLE SPACED
12   MOVE C'VSAM RC =' TO PRT1-9
     MOVE VAL247-247-B TO PRT11-13 0
     PRINT
     MOVE C'VSAM EC =' TO PRT1-9
     MOVE VAL248-248-B TO PRT11-13 0
     PRINT
     MOVE C'VAL253-255 =' TO PRT1-12
     MOVE VAL253-255 TO PRT14
     PRINT
     MOVE C'0016' TO VAL46-49
     GO TO 14
    *-------------------*
    *  SUCCESSFUL TEST  *
    *-------------------*
13   MOVE C'END OF FILE INF' TO PRT1-15
     PRINT DOUBLE SPACED
14   MOVE C'END OF TEST' TO PRT1-11
     PRINT TRIPLE SPACED
     GO TO EOJ
9999END
```

Example 33

Native Variable Length VSAM (KSDS, ESDS) Using OPEN/CLOSE, GET, WRITE, SET PTA, READ, SETGENKEY, ONERROR

This example contains variable length VSAM files, ESDS and KSDS, utilizing native VSAM. Both the ESDS and KSDS files are loaded, then read sequentially and randomly using READ and SETGENKEY with GET. A print trail, shows the flow of the program through its various stages. Note that files INF and OFA, and DET and OFB are the same.

Prior to running the VISION:Report, an IDCAMS job similar to the following was run (although the IDCAMS is a MVS job stream, most, if not all the IDCAMS statements, would be identical under VSE).

```
DELETE   ESDS.TEST     CLUSTER
DELETE   KSDS.TEST     CLUSTER
SET MAXCC=0
DEFINE CLUSTER (NAME(KSDS.TEST) -
                VOLUMES(volser) -
                RECORDSIZE(10 90) -
                KEYS(10 0) -
                INDEXED) -
       DATA    (NAME(KSDS.TEST.DATA) -
                CISZ(512) -
                TRACKS(1 1)) -
       INDEX   (NAME(KSDS.TEST.INDEX))
DEFINE CLUSTER (NAME(ESDS.TEST) -
                VOLUMES(volser) -
                RECORDSIZE(10 90) -
                NONINDEXED) -
       DATA    (NAME(ESDS.TEST.DATA) -
                CISZ(512) -
                TRACKS(1 1))
```

VSE JCL Example

```
// JOB SAMP33
// DLBL filename,'your.VISION.lib'
// EXTENT   ,volser
// LIBDEF  PHASE,SEARCH=(lib.sublib)
// DLBL     INF,'ESDS.TEST',,VSAM
// DLBL     OFA,'ESDS.TEST',,VSAM
// DLBL     DET,'KSDS.TEST',,VSAM
// DLBL     OFB,'KSDS.TEST',,VSAM
// EXEC  QUKBJOB
  ...  VISION:Report statements as shown below
/*
/&
```

MVS JCL Example

```
//SAMP33   JOB  (800-0000,0000),'Example 34'
//STEP1    EXEC  QJTEST
//QJ.SYSUT1   DD  DISP=SHR,DSN=ESDS.TEST         INF
//QJ.SYSUT2   DD  DISP=SHR,DSN=ESDS.TEST         OFA
//QJ.SYSDET   DD  DISP=SHR,DSN=KSDS.TEST         DET
//QJ.SYSUT3   DD  DISP=SHR,DSN=KSDS.TEST         OFB
//QJ.SYSIN    DD  *
  ...  VISION:Report statements as shown below
/*
//
```

VISION:Report Statements

```
*-------------------------------------------------------*
* SAMP33:   VARIABLE LENGTH VSAM, KSDS & ESDS           *
*           USING NATIVE VSAM.                          *
*-------------------------------------------------------*
*  OUTPUT FILES OFC AND OFD ARE NOT USED.               *
*  INPUT FILES INC AND IND ARE NOT USED.                *
*                                                       *
*  FILE OFA IS A VSAM ESDS CLUSTER.                     *
*  FILE OFB IS A VSAM KSDS CLUSTER.                     *
*                                                       *
*  FILE INF IS THE SAME DATASET AS FILE OFA.            *
*  FILE DET IS THE SAME DATASET AS FILE OFB.            *
*                                                       *
*-------------------------------------------------------*
*                                                       *
*  1.  LOAD ESDS AND KSDS FILES, USING WRITE            *
*  2.  GET ESDS AND KSDS FILES SEQUENTIALLY, USING GET*
*  3.  READ ESDS/KSDS FILES RANDOMLY, USING READ        *
*  4.  READ ESDS/KSDS FILES RANDOMLY, USING SETGENKEY *
*                                                       *
*  THERE ARE OPEN/CLOSE AND SET PTX THROUGHOUT PROGRAM*
*  ONERROR OPTION USED ON SOME VSAM VERBS.              *
*                                                       *
*-------------------------------------------------------*
*INFESDS    0090                   /* If VSE, remove * at position 1
*OFAESDS    0090                   /* If VSE, remove * at position 1
*DETKSDS    0090                   /* If VSE, remove * at position 1
*OFBKSDS    0090                   /* If VSE, remove * at position 1
     EQU RECORD    WST1-7    C'RECORD '
     EQU RECNO     WST8      C'0'
     EQU FILLER-1  WST9-10   SPACES
     EQU TYPE      WST11-14  SPACES
     EQU FILLER-2  WST15-20  C'  ..2'
     EQU FILLER-3  WST21-30  C'....+....3'
     EQU FILLER-4  WST31-40  C'....+....4'
     EQU FILLER-5  WST41-50  C'....+....5'
     EQU FILLER-6  WST51-60  C'....+....6'
     EQU FILLER-7  WST61-70  C'....+....7'
     EQU FILLER-8  WST71-80  C'....+....8'
     EQU RBA       (4)-B
     EQU WST-KEY   (10)
     EQU LENGTH    (2)-B     ZERO
     EQU VERB      (9)       SPACES
     EQU FILE      (3)       SPACES
     TITLE '$IPLDAT$          '
     TITLE 'LOAD AND READ '
     TITLE 'VARIABLE LENGTH ESDS AND KSDS'
     TITLE '          PAGE $PG$'
```

```
    *---------------------------------*
    *  LOAD VARIABLE LENGTH ESDS/KSDS  *
    *---------------------------------*
     MOVE C'LOAD VARIABLE LENGTH VSAM CLUSTERS' TO PRT1
     PRINT
     PRINT
     MOVE C'WRITE    ' TO VERB
1    ADD C'10' TO LENGTH
     ADD C'1' TO RECNO
     MOVE C'ESDS' TO TYPE
     MOVE C'OFA' TO FILE
     SET PTA OFA1
     SET PTA DOWN 2
     MOVE LENGTH TO PTA1-2-B
     MOVE RECORD TO OFA1 LENGTH
     WRITE OFA ONERROR 5
     MOVE C'KSDS' TO TYPE
     MOVE C'OFB' TO FILE
     SET PTA OFB1
     SET PTA DOWN 2
     MOVE LENGTH TO PTA1-2-B
     MOVE RECORD TO OFB1 LENGTH
     WRITE OFB ONERROR 5
     MOVE C' SDS' TO TYPE
     MOVE RECORD TO PRT1 LENGTH
     PRINT
     IF LENGTH IS LT  X'0050'            /* If record length < 80
       GO TO 1.
2    MOVE C'CLOSE FILES OFA AND OFB' TO PRT1
     PRINT DOUBLE SPACED
     CLOSE OFA
     CLOSE OFB
    *-----------------------------------------------*
    *  READ ESDS/KSDS SEQUENTIALLY AND ECHO PRINT  *
    *-----------------------------------------------*
     MOVE C'OPEN FILES INF AND DET' TO PRT1
     PRINT DOUBLE SPACED
     OPEN INF
     OPEN DET
     MOVE C'GET ESDS/KSDS SEQUENTIALLY' TO PRT1
     PRINT DOUBLE SPACED
     PRINT
     MOVE C'GET      ' TO VERB
     MOVE C'DET' TO FILE
3   GET INF ATEND 4
          SET PTA INF1
          SET PTA DOWN 2
          MOVE INF1 TO PRT1 PTA1-2-B
          PRINT
          GET DET ATEND 5
          SET PTA DET1
          SET PTA DOWN 2
          MOVE DET1 TO PRT1 PTA1-2-B
          PRINT
          PRINT
          GO TO 3
4   MOVE C'END OF FILE INF' TO PRT1-15
          PRINT DOUBLE SPACED
          MOVE C'CLOSE FILES INF AND DET' TO PRT1
          PRINT DOUBLE SPACED
          CLOSE INF
          CLOSE DET
    *--------------------------------------*
    *  READ VSAM ESDS RANDOMLY USING READ  *
    *--------------------------------------*
```

```
 MOVE C'OPEN FILES INF AND DET' TO PRT1
 PRINT DOUBLE SPACED
 OPEN INF
 OPEN DET
 MOVE C'READING VSAM ESDS RANDOMLY' TO PRT1-26
 MOVE C'USING READ' TO PRT28-41
 PRINT DOUBLE SPACED
 MOVE C'RBA = 60' TO PRT1
 PRINT
 MOVE P'60' TO RBA
 MOVE C'READ     ' TO VERB
 MOVE C'INF' TO FILE
 READ INF USING RBA EQUAL ONERROR 5
 MOVE C'RANDOM READ WORKED' TO PRT1-18
 PRINT
 SET PTA INF1
 SET PTA DOWN 2
 MOVE INF1 TO PRT1 PTA1-2-B
 PRINT DOUBLE SPACED
*-------------------------------------*
*  READ VSAM KSDS RANDOMLY USING READ  *
*-------------------------------------*
 MOVE C'READING VSAM KSDS RANDOMLY' TO PRT1-26
 MOVE C'USING READ' TO PRT28-41
 PRINT DOUBLE SPACED
 MOVE C'SEARCH KEY = RECORD 4  ' TO PRT1
 PRINT
 MOVE C'RECORD 4  ' TO WST-KEY
 MOVE C'READ     ' TO VERB
 MOVE C'DET' TO FILE
 READ DET USING WST-KEY EQUAL ONERROR 5
 MOVE C'RANDOM READ WORKED' TO PRT1-18
 PRINT
 SET PTA DET1
 SET PTA DOWN 2
 MOVE DET1 TO PRT1 PTA1-2-B
 PRINT DOUBLE SPACED
*-----------------------------------------*
*  READ VSAM KSDS RANDOMLY USING SETGENKEY  *
*-----------------------------------------*
 MOVE C'READING VSAM KSDS RANDOMLY' TO PRT1-26
 MOVE C'USING SETGENKEY AND GET' TO PRT28
 PRINT DOUBLE SPACED
 MOVE C'SEARCH KEY = RECORD 2  ' TO PRT1-23
 PRINT
 MOVE C'RECORD 2  ' TO WST-KEY
 MOVE C'SETGENKEY' TO VERB
 MOVE C'DET' TO FILE
 SETGENKEY DET USING WST-KEY EQUAL ONERROR 5
 MOVE C'GET      ' TO VERB
 GET DET ATEND 5
 MOVE C'RANDOM READ WORKED' TO PRT1-18
 PRINT
 SET PTA DET1
 SET PTA DOWN 2
 MOVE DET1 TO PRT1 PTA1-2-B
 PRINT DOUBLE SPACED
 GET DET ATEND 5
 MOVE DET1 TO PRT1 PTA1-2-B
 PRINT
*-----------------------------------------*
*  READ VSAM ESDS RANDOMLY USING SETGENKEY  *
*-----------------------------------------*
```

```
     MOVE C'READING VSAM ESDS RANDOMLY' TO PRT1-26
     MOVE C'USING SETGENKEY AND GET' TO PRT28
     PRINT DOUBLE SPACED
     MOVE C'RBA = 10' TO PRT1
     PRINT
     MOVE P'10' TO RBA
     MOVE C'SETGENKEY' TO VERB
     MOVE C'INF' TO FILE
     SETGENKEY INF USING RBA EQUAL ONERROR 5
     MOVE C'GET      ' TO VERB
     GET INF ATEND 5
     MOVE C'RANDOM READ WORKED' TO PRT1-18
     PRINT
     SET PTA INF1
     SET PTA DOWN 2
     MOVE INF1 TO PRT1 PTA1-2-B
     PRINT DOUBLE SPACED
     GET INF ATEND 5
     MOVE INF1 TO PRT1 PTA1-2-B
     PRINT
     MOVE C'END OF TEST' TO PRT1-11
     PRINT TRIPLE SPACED
     GO TO EOJ
    *--------------*
    *  VSAM ERROR  *
    *--------------*
5    MOVE C'***  VSAM ERROR  ***' TO PRT1
     PRINT TRIPLE SPACED
     MOVE C'FILE =' TO PRT1-6
     MOVE FILE TO PRT8
     PRINT
     MOVE C'VERB =' TO PRT1-6
     MOVE VERB TO PRT8
     PRINT
     MOVE C'VAL253-255 =' TO PRT1-12
     MOVE VAL253-255 TO PRT14
     PRINT
     MOVE C'VSAM RC =' TO PRT1-9
     MOVE VAL247-247-B TO PRT11-13 0
     PRINT
     MOVE C'VSAM EC =' TO PRT1-9
     MOVE VAL248-248-B TO PRT11-13 0
     PRINT
     MOVE C'TEST TERMINATED' TO PRT1
     PRINT TRIPLE SPACED
     MOVE C'0016' TO VAL46-49
     GO TO EOJ
9999 END
```

Example 34

QUIKVSAM (KSDS) with Alternate Index, Using OPTION, OPEN/CLOSE, LOAD, READ, GET-UPD, READ-UPD, ADD, GET, POINT, UPDATE, ERASE

This example is very similar to Example 32. The primary differences between these two examples are that in this example, the VSAM file is a KSDS, and QUIKVSAM is used, rather than native VSAM.

Prior to running the VISION:Report, an IDCAMS job similar to the following was run (although the IDCAMS is a MVS job stream, most, if not all the IDCAMS statements, would be identical under VSE).

```
DELETE KSDS.TEST CLUSTER
  SET MAXCC=0
  DEFINE   CLUSTER (NAME(KSDS.TEST) -
                    VOLUMES(volser) -
                    RECORDSIZE(80 80) -
                    KEYS(20 10) -
                    INDEXED) -
           DATA    (NAME(KSDS.TEST.DATA) -
                    CISZ(512) -
                    TRACKS(1 1)) -
           INDEX   (NAME(KSDS.TEST.INDEX))
  REPRO    INFILE(SYSUT1) -
           OUTFILE(KSDS.TEST)
  PRINT    INFILE(KSDS.TEST) -
           CHARACTER
  DEFINE   AIX     (NAME(KSDS.TEST.AIX) -
                    RELATE(KSDS.TEST) -
                    VOLUMES(volser) -
                    KEYS(10 0) -
                    RECSZ(128 256) -
                    TRACKS(1 1) -
                    UPGRADE) -
           DATA    (NAME(KSDS.TEST.AIX.DATA) -
                    CISZ(1024)) -
           INDEX   (NAME(KSDS.TEST.AIX.INDEX) -
                    CISZ(512))
  DEFINE   PATH    (NAME(KSDS.TEST.PATH) -
                    PATHENTRY(KSDS.TEST.AIX) -
                    UPDATE)
  BLDINDEX INDATASET(KSDS.TEST) -
           OUTDATASET(KSDS.TEST.AIX) –
           INTERNALSORT
```

The input to build the KSDS file is as follows:

```
EEEEEEEEEETEST RECORD NUMBER 1
DDDDDDDDDDTEST RECORD NUMBER 2
CCCCCCCCCCTEST RECORD NUMBER 3
BBBBBBBBBBTEST RECORD NUMBER 4
AAAAAAAAAATEST RECORD NUMBER 5
```

VSE JCL Example

```
// JOB SAMP34
// DLBL filename,'your.VISION.lib'
// EXTENT   ,volser
// LIBDEF  PHASE,SEARCH=(lib.sublib)
// DLBL     OFA,'KSDS.TEST.PATH',,VSAM
// DLBL     OFB,'KSDS.TEST',,VSAM
// EXEC  QUKBJOB,SIZE=512K
  ...  VISION:Report statements as shown below
/*
/&
```

MVS JCL Example

```
//SAMP34   JOB  (800-0000,0000),'Example 35'
//STEP1   EXEC  QJTEST
//QJ.SYSUT2   DD  DSNAME=KSDS.TEST.PATH,DISP=SHR
//QJ.SYSUT3   DD  DSNAME=KSDS.TEST,DISP=SHR
//QJ.SYSIN    DD  *
  ...  VISION:Report statements as shown below
/*
//
```

VISION:Report Statements

```
OPTION SPIE=NO                    /* MVS only
*-------------------------------------------------------*
* SAMP34:   VSAM KSDS WITH ALTERNATE INDEX              *
*           USING QUIKVSAM.                             *
*-------------------------------------------------------*
* VERBS USED:                                           *
*  OPEN                   GET                          *
*  CLOSE                  OPTION                       *
*  READ                   POINT                        *
*  GET-UPD                UPDATE                       *
*  READ-UPD               ERASE                        *
*  ADD                    LOAD                         *
*-------------------------------------------------------*
*  OUTPUT FILES OFA, OFB, OFC, AND OFD ARE NOT USED.    *
*  INPUT FILES INF, DET, INC, AND IND ARE NOT USED.     *
*                                                       *
*  //SYSUT3 DD IS A NONEMPTY VSAM KSDS CLUSTER. (OFB)   *
*                                                       *
*  //SYSUT2 DD IS A PATH FOR AN ALTERNATE INDEX (OFA)   *
*  TO THE //SYSUT3 DD CLUSTER.                          *
*-------------------------------------------------------*
     EQU WST-KEY  WST1-10
     TRACE LAST5D
     TITLE '$IPLDAT$           QUIKVSAM KSDS ALTERNATE INDEX '
     TITLE 'TEST          PAGE $PG$'
    *---------------------------------------*
    *  VSAM KSDS GET USING ALTERNATE INDEX  *
    *---------------------------------------*
```

```
     MOVE C'QUIKVSAM OPEN ALTERNATE INDEX PATH' TO PRT1
     PRINT
     MOVE LOVALUE TO SAV1-13
     MOVE SPACES TO SAV14-22
     CALL QUIKVSAM  C'SYSUT2  '  C'OPEN'  SAV1
     IF SAV8-9 NOT = LOVALUE
       MOVE C'OPEN' TO SAV14
       GO TO 5.
     MOVE C'QUIKVSAM GET' TO PRT1-12
     MOVE C'USING ALTERNATE INDEX' TO PRT14
     PRINT DOUBLE SPACED
     PRINT
1    CALL QUIKVSAM  C'SYSUT2  '  C'GET'  PRT1
     IF PRT1-10 = HIVALUE
       GO TO 2.
     IF SAV8-9 NOT = LOVALUE
       MOVE C'GET' TO SAV14
       GO TO 5.
     PRINT
     GO TO 1
2    MOVE SPACES TO PRT1-120
     MOVE C'END OF FILE SYSUT2' TO PRT1
     PRINT DOUBLE SPACED
     MOVE C'QUIKVSAM CLOSE' TO PRT1
     PRINT DOUBLE SPACED
     CALL QUIKVSAM  C'SYSUT2  '  C'CLOSE'
     IF SAV8-9 NOT = LOVALUE
       MOVE C'CLOSE' TO SAV14
       GO TO 5.
    *----------------------------------------*
    *  VSAM KSDS READ USING ALTERNATE INDEX  *
    *----------------------------------------*
     MOVE C'QUIKVSAM OPTION ALTERNATE INDEX PATH' TO PRT1
     PRINT DOUBLE SPACED
     MOVE LOVALUE TO SAV1-13
     CALL QUIKVSAM  C'SYSUT2  '  C'OPTION'  SAV1
     IF SAV8-9 NOT = LOVALUE
       MOVE C'OPTION' TO SAV14
       GO TO 5.
     MOVE C'QUIKVSAM READ' TO PRT1-13
     MOVE C'USING ALTERNATE INDEX' TO PRT15
     PRINT DOUBLE SPACED
     MOVE C'SEARCH KEY = CCCCCCCCCC' TO PRT1-23
     PRINT
     MOVE C'CCCCCCCCCC' TO WST-KEY
     CALL QUIKVSAM  C'SYSUT2  '  C'READ'  PRT1  WST1  C'KEQ'
     IF SAV8-9 NOT = LOVALUE
       MOVE C'READ' TO SAV14
       GO TO 5.
     PRINT DOUBLE SPACED
    *----------------------------------------*
    *  VSAM KSDS POINT USING ALTERNATE INDEX  *
    *----------------------------------------*
```

```
 MOVE C'QUIKVSAM POINT' TO PRT1-14
 MOVE C'USING ALTERNATE INDEX' TO PRT16
 PRINT DOUBLE SPACED
 MOVE C'SEARCH KEY = BBBBBBBBBB' TO PRT1-23
 PRINT
 MOVE C'BBBBBBBBBB' TO WST-KEY
 CALL QUIKVSAM  C'SYSUT2  '  C'POINT'  WST1  C'KEQ'
 IF SAV8-9 NOT = LOVALUE
   MOVE C'POINT' TO SAV14
   GO TO 5.
 MOVE C'QUIKVSAM GET' TO PRT1-12
 MOVE C'USING ALTERNATE INDEX' TO PRT14
 PRINT DOUBLE SPACED
 CALL QUIKVSAM  C'SYSUT2  '  C'GET'  PRT1
 IF SAV8-9 NOT = LOVALUE
   MOVE C'GET' TO SAV14
   GO TO 5.
 PRINT DOUBLE SPACED
 MOVE C'QUIKVSAM GET-UPD' TO PRT1-16
 MOVE C'USING ALTERNATE INDEX' TO PRT18
 PRINT DOUBLE SPACED
 CALL QUIKVSAM  C'SYSUT2  '  C'GET-UPD'  WST11
 IF SAV8-9 NOT = LOVALUE
   MOVE C'GET-UPD' TO SAV14
   GO TO 5.
 MOVE WST11-50 TO PRT1-40
 PRINT DOUBLE SPACED
*----------------------------------------*
*  VSAM KSDS UPDATE USING ALTERNATE INDEX  *
*----------------------------------------*
 MOVE C'QUIKVSAM UPDATE' TO PRT1-15
 MOVE C'USING ALTERNATE INDEX' TO PRT17
 PRINT DOUBLE SPACED
 MOVE C' UPDATE   ' TO WST41
 MOVE WST11-50 TO PRT1
 PRINT DOUBLE SPACED
 CALL QUIKVSAM  C'SYSUT2  '  C'UPDATE'  WST11
 IF SAV8-9 NOT = LOVALUE
   MOVE C'UPDATE' TO SAV14
   GO TO 5.
*---------------------------------------*
*  VSAM KSDS ERASE USING ALTERNATE INDEX  *
*---------------------------------------*
 MOVE C'QUIKVSAM READ-UPD' TO PRT1-17
 MOVE C'USING ALTERNATE INDEX' TO PRT19
 PRINT DOUBLE SPACED
 MOVE C'SEARCH KEY = BBBBBBBBBB' TO PRT1-23
 PRINT
 MOVE C'BBBBBBBBBB' TO WST-KEY
 CALL QUIKVSAM  C'SYSUT2  '  C'READ-UPD'  PRT1  WST1  C'KEQ'
 IF SAV8-9 NOT = LOVALUE
   MOVE C'READ-UPD' TO SAV14
   GO TO 5.
 PRINT DOUBLE SPACED
 MOVE C'QUIKVSAM ERASE' TO PRT1-14
 MOVE C'USING ALTERNATE INDEX' TO PRT16
 PRINT DOUBLE SPACED
 CALL QUIKVSAM  C'SYSUT2  '  C'ERASE'
 IF SAV8-9 NOT = LOVALUE
   MOVE C'ERASE' TO SAV14
   GO TO 5.
*--------------------------------------*
*  VSAM KSDS ADD USING ALTERNATE INDEX  *
*--------------------------------------*
```

```
      MOVE C'QUIKVSAM ADD' TO PRT1-12
      MOVE C'USING ALTERNATE INDEX' TO PRT14
      PRINT DOUBLE SPACED
      MOVE C'D 9' TO WST11-20
      MOVE C'TEST RECORD NUMBER 0 ADD      ' TO WST21
      MOVE P'40' TO SAV4-7-B
      MOVE WST11-50 TO PRT1
      PRINT DOUBLE SPACED
      CALL QUIKVSAM  C'SYSUT2  '  C'ADD'  WST11
      IF SAV8-9 NOT = LOVALUE
        MOVE C'ADD' TO SAV14
        GO TO 5.
     *---------------------------------------*
     *  VSAM KSDS LOAD USING ALTERNATE INDEX  *
     *---------------------------------------*
      MOVE C'QUIKVSAM LOAD' TO PRT1-13
      MOVE C'USING ALTERNATE INDEX' TO PRT15
      PRINT DOUBLE SPACED
      MOVE C'# 9' TO WST11-20
      MOVE C'TEST RECORD NUMBER 6' TO WST21-40
      MOVE C' LOAD     ' TO WST41-50
      MOVE P'40' TO SAV4-7-B
      MOVE WST11-50 TO PRT1-40
      PRINT DOUBLE SPACED
      CALL QUIKVSAM  C'SYSUT2  '  C'LOAD'  WST11
      IF SAV8-9 NOT = LOVALUE
        MOVE C'LOAD' TO SAV14
        GO TO 5.
      MOVE C'QUIKVSAM CLOSE' TO PRT1-14
      MOVE C'ALTERNATE INDEX PATH' TO PRT16
      PRINT DOUBLE SPACED
      CALL QUIKVSAM  C'SYSUT2  '  C'CLOSE'
      IF SAV8-9 NOT = LOVALUE
        MOVE C'CLOSE' TO SAV14
        GO TO 5.
     *-----------------------------------*
     *  VSAM KSDS GET USING PRIMARY KEY  *
     *-----------------------------------*
      MOVE C'QUIKVSAM OPEN BASE CLUSTER' TO PRT1
      PRINT DOUBLE SPACED
      MOVE LOVALUE TO SAV1-13
      CALL QUIKVSAM  C'SYSUT3  '  C'OPEN'  SAV1
      IF SAV8-9 NOT = LOVALUE
        MOVE C'OPEN' TO SAV14
        GO TO 5.
      MOVE C'QUIKVSAM GET' TO PRT1-12
      MOVE C'USING PRIMARY INDEX' TO PRT14
      PRINT DOUBLE SPACED
      PRINT
3     CALL QUIKVSAM  C'SYSUT3  '  C'GET'  PRT1
      IF PRT1-10 = HIVALUE
        GO TO 4.
      IF SAV8-9 NOT = LOVALUE
        MOVE C'GET' TO SAV14
        GO TO 5.
      PRINT
      GO TO 3
```

```
4    MOVE SPACES TO PRT1-120
     MOVE C'END OF FILE SYSUT3' TO PRT1
     PRINT DOUBLE SPACED
     MOVE C'QUIKVSAM CLOSE' TO PRT1
     PRINT DOUBLE SPACED
     CALL QUIKVSAM  C'CLOSE'
     IF SAV8-9 NOT = LOVALUE
       MOVE C'CLOSE' TO SAV14
       GO TO 5.
     MOVE C'END OF TEST' TO PRT1-11
     PRINT TRIPLE SPACED
     MOVE ZERO TO VAL46-49
     GO TO EOJ
    *-----------------*
    *  QUIKVSAM ERROR  *
    *-----------------*
5    MOVE SPACES TO PRT1-120
     MOVE C'QUIKVSAM FAILED.  FUNCTION =' TO PRT1-28
     MOVE SAV14-22 TO PRT30
     PRINT DOUBLE SPACED
     MOVE C'COMMUNICATION AREA DUMP:' TO PRT1
     PRINT DOUBLE SPACED
     PRINTHEX SAV1-13
     MOVE C'TEST TERMINATED' TO PRT1
     PRINT TRIPLE SPACED
     GO TO EOJ
9999 END
```

Example 35

Troubleshooting Problems

The following is an example of a program that utilizes the debugging facilities of VISION:Report, with the one statement that is different between VSE and MVS. See the section Troubleshooting and Memory Requirements in Chapter 5 for a more detailed information on troubleshooting, including suppression of ABEND-AID if you have this product.

VSE Statement

```
OPTION  STXITPC=NO    /* Turn off VISION:Report program check-VSE only
.. VISION:Report Statements as shown below
```

MVS Statement

```
OPTION  SPIE=NO       /* Turn off VISION:Report program check-MVS only
.. VISION:Report Statements as shown below
```

VISION:Report Statements

```
OPTION  TRACECT=1     /* Forces printing sequence numbers immed.
OPTION  LISTOPT=YES   /* Show what options running under
OPTION  LIST=YES      /* Ensure that VISION:Report statements print
*
* THE FOLLOWING "TRACE" STATEMENT HAS SEVERAL OPTIONS:
*
*     ALL    - TRACES EVERY STATEMENT THAT VISION:REPORT
*              EXECUTES. WHEN THE TRACECT HAS BEEN REACHED,
*              THE VISION:REPORT SEQUENCE NUMBER IS PRINTED.
*              NOTE:  THE AMOUNT OF PRINTING MAY BE EXCESSIVE,
*                     BUT SOMETIMES, THIS MAY BE THE ONLY
*                     METHOD. IF YOU KNOW WHERE THE PROBLEM
*                     EXISTS, YOU MAY PUT IN A TRACE STATEMENT
*                     THERE.
*     LAST50 - CAUSES MEMORY TABLE OF VISION:REPORT INTERNAL
*              SEQUENCE NUMBERS TO BE CREATED AND POSTED.
*              NO TRACE PRINTING OCCURS ON THE PRINTER
*              UNLESS THE PROGRAM ABORTS DUE TO A PROGRAM
*              CHECK.
*              NOTE:  SOMETIMES THE TRACE TABLE MAY NOT BE
*                     PRINTED, DEPENDING UPON THE TYPE OF
*                     ABEND.
*
*     OFF    - TURNS OFF THE TRACE ROUTINE.
*
TRACE ALL                   /* Trace last50 may not do it!!
..
..
  GOTO EOJ
9999 END
```

Example 36

Mixture of Native VSAM and CALL to QUIKVSAM, with Field Names Greater Than 14 Characters, and Forcing $PAGE$ to be Greater Than 6 Digits

The VISION:Report program would be similar to the following:

```
OPTION SRTSIZE=1024   /*  ===> SRTSIZE=1024 IF MVS, SORTSIZ IF VSE
*INMDISC52800352SSYS005      /* IF VSE, REMOVE * AT POSITION 1
*OFFVSAM                     /* IF VSE, REMOVE * AT POSITION 1
    *-------------------------------------------------------*
    * SAMP36:                                               *
    *                                                       *
    *  TEST OF VARIOUS 16.0+ NEW FEATURES                   *
    *  ----------------------------------                   *
    *  1.  MORE THAN 14 CHARACTERS FOR A FIELD-NAME.        *
    *      THIS APPEARS ON THE "EQU" AND "REPORT" VERB.     *
    *  2.  NATIVE VSAM AND CALLING QUIKVSAM WORKS.          *
    *  3.  $PAGE$ RESERVED WORD, ALLOWING FOR               *
    *      MORE THAN 6 DIGITS PAGE NUMBER.                  *
    *      TEST IT BY FORCING PNR1-4-P TO BE 999998.        *
    *-------------------------------------------------------*
    *  LOGIC:                                               *
    *                                                       *
    *  RUN IDCAMS JOB TO DEFINE VSAM DATASET BEFORE         *
    *  RUNNING THIS VISION:REPORT JOB.                      *
    *                                                       *
    *                                                       *
    *  SORT INPUT FILE INM BY ACCT-CODE (182-183) MAJOR     *
    *  AND ACCOUNT NUMBER (4-10).                           *
    *                                                       *
    *  CREATE OUTPUT FILE, OFF, FROM INPUT FILE INM.        *
    *  USE "OPEN'S" ON BOTH FILES.                          *
    *                                                       *
    *  TITLE CARD HAS "$PAGE$.                              *
    *  MANUALLY MOVE KEY TO PRINT AREA AND PRINT IT.        *
    *                                                       *
    *  AT END OF FILE FOR INM, CLOSE INM AND OFF FILES.     *
    *                                                       *
    *  CALL QUIKVSAM TO PROCESS INPUT FILE, SYSINN.         *
    *  THIS IS REALLY THE ORIGINAL OUTPUT FILE, OFF.        *
    *  WE WILL OPEN, GET, AND CLOSE FILE SYSINN.            *
    *                                                       *
    *  FORCE PAGE NUMBER (PNR1-4-P) TO P'999998'.           *
    *  USE "PRINT REPORT" VERB TO DO REPORT.                *
    *  WE SHOULD SEE ROLL-OVER OF PAGE NUMBERS FROM         *
    *  999998 TO 1,000,002 OR THEREABOUTS.                  *
    *                                                       *
    *-------------------------------------------------------*
****************************************************************
*                                                              *
*  NOTE:  THE FOLLOWING IDCAMS JOB WILL NEED TO BE RUN         *
*         PRIOR TO EXECUTION OF THIS JOB.                      *
*                                                              *
*         CHANGE THE ISPQJ.SAMP36 TO YOUR COMPANY'S            *
*         STANDARDS, AS WELL AS THE 'VOLSER'.                  *
*                                                              *
****************************************************************
*--------------------------------------------------------------*
*                                                               *
*      DELETE (ISPQJ.SAMP36.ARFILE.VSAM) CLUSTER                *
```

```
*      SET MAXCC=0                                                    *
*      DEFINE CLUSTER (NAME (ISPQJ.SAMP36.ARFILE.VSAM)       -        *
*             VOL (VOLSER)                                   -        *
*             RECSZ(352 352)   KEY (9,210) )                 -        *
*             DATA ( NAME (ISPQJ.SAMP36.ARFILE.VSAM.DATA)    -        *
*             SPEED                                          -        *
*             TRK (3,1) FREESPACE (20,5) )                   -        *
*         INDEX ( NAME (ISPQJ.SAMP36.ARFILE.VSAM.INDEX) )              *
*                                                                      *
*----------------------------------------------------------------------*
EQU  THIS-IS-A-FULL-LENGTH-KEY    WST1-20        /* 16.0+ ONLY
     SORT FILE INM ON INM182-183  INM4-10
     TITLE 'TEST OF PAGE    $PAGES$ '           /* 16.0+ ONLY
* NOTE:  $PAGES$ WILL ALLOW A 7-BYTE PAGE NUMBER !!!!
     REPORT  THIS-IS-A-FULL-LENGTH-KEY  (KEY)    /* 16.0 + ONLY
             WST11-36   (NAME)
   TRACE LAST50
     OPEN OFF
     OPEN INM
     SET PTA OFF1
     SET PTA DOWN 2
     MOVE P'352'         TO  PTA1-2-B
     MOVE  SPACES        TO  OFF1-352

010  GET  INM  ATEND 300
     MOVE  INM1-210      TO  OFF1-210
* BUILD KEY WITH NEXT 2 INSTRUCTIONS
     MOVE  INM182-183    TO  OFF211-212        /* ACCOUNT-CODE
     MOVE  INM4-10       TO  OFF213-219        /* ACCT
     MOVE  INM220-352    TO  OFF220-352        /* REST OF IT

     MOVE  OFF211-219    TO  PRT1
     MOVE  INM85-109     TO  PRT30
     PRINT
     WRITE OFF   ONERROR   200
     GOTO 010

200  MOVE  C'====> ERROR DURING LOADING'    TO PRT20
     MOVE  OFF211-219    TO  PRT1
     PRINT
     GOTO 010

300  MOVE  C'END OF LOAD PHASE'    TO PRT1
     PRINT
     MOVE  C'SEQUENTIAL RETRIEVAL-USING QUIKVSAM'    TO PRT1
     PRINT
* * * * * * * * * *
     MOVE P'999998' TO PNR1-4-P   /*    16.0+ ONLY
*                                 /* FORCE TO HIGH NUMBER,
* * * * * * * * * * *              /*   SO WE CAN CHECK 7-DIGITS
     MOVE  C'PAGE NUMBER'    TO PRT1
     PRINT
     CLOSE INM
     MOVE  C'CLOSE INM'    TO PRT1
     PRINT
     CLOSE OFF
     MOVE  C'CLOSE OFF'    TO PRT1
     PRINT
* NOTE:  MIXING NATIVE VSAM WITH QUIKVSAM         /* 16.0+ ONLY
     CALL QUIKVSAM C'SYSINN  ' C'OPEN'  SAV1
*
     MOVE  C'CALL QUIKVSAM W/SYSINN'  TO PRT1
     PRINT
     IF SAV8-9 NOT = LOVALUE
       MOVE C'OPEN ERROR' TO PRT1
```

```
      PRINT
      GO TO EOJ.
320
     CALL QUIKVSAM  C'SYSINN  ' C'GET '  WST1
     IF WST1-10 = HIVALUE
       GO TO 400.
     IF SAV8-9 NOT = LOVALUE
       MOVE C'GET ERROR' TO SAV14
       PRINT
       GO TO EOJ.
     PRINT REPORT
     GOTO 320
400
     MOVE C'END OF FILE USING QUIKVSAM SEQ RETRIEVAL' TO PRT1
     PRINT DOUBLE SPACED
     MOVE C'QUIKVSAM CLOSE' TO PRT1
     PRINT DOUBLE SPACED
     CALL QUIKVSAM  C'SYSINN  ' C'CLOSE'  SAV1
     GOTO  EOJ
9999END
```

Example 37

Various Usages of IF (Nested IF, IF with Parentheses, IF/ELSE/ENDIF) and Bit Manipulation Instructions (such as AND, OR, XOR, TRAN, TRNT)

The VISION:Report program would be similar to the following:

```
OPTION SEQCHK=NO
OPTION IFNUM=YES
OPTION TRACECT=1,U336DMP=YES

* ************************************************************
*                                                            *
* SAMP37:                                                    *
*         TESTING OF FOLLOWING FUNCTIONS:                    *
*                                                            *
*            OR, AND, XOR, TRAN, TRNT                        *
*            IF ... W/NESTED IF                              *
*                   PARENTHESES                              *
*                   ELSE/ENDIF                               *
*                                                            *
* NOTE: ====> RUNS ONLY UNDER RELEASE 16.0+ !!!! <======    *
*                                                            *
* ************************************************************
EQU ALL-FIELDS       WST0
* FOR OR INSTRUCTION
EQU OR-FLDA          (6)   C'ABCDEF'
EQU OR-FLDB          (6)   X'303030303030'

EQU AND-FLDA         (6)   C'ABCDEF'
EQU AND-FLDB         (6)   X'BFBFBFBFBFBF'

EQU XOR-FLDA         (6)   X'304050607080'
EQU XOR-FLDB         (6)   X'303030303030'

EQU TRAN-FLDA        (20)  C'BLUEJAYS EAT PEANUTS'
EQU TRAN-FLDB        (256)
EQU TRAN-FLDB             /* REDEFINES
*                        0 1 2 3 4 5 6 7 8 9 A B C D E F      *
*
EQU FILLER     (16)  X'FFFEFDFCFBFAF9F8F7F6F5F4F3F2F1F0'  /* 0
EQU FILLER     (16)  X'EFEEEDECEBEAE9E8E7E6E5E4E3E2E1E0'  /* 1
EQU FILLER     (16)  X'DFDEDDDCDBDAD9D8D7D6D5D4D3D2D1D0'  /* 2
EQU FILLER     (16)  X'CFCECDCCCBCAC9C8C7C6C5C4C3C2C1C0'  /* 3
EQU FILLER     (16)  X'BFBEBDBCBBBAB9B8B7B6B5B4B3B2B1B0'  /* 4
EQU FILLER     (16)  X'AFAEADACABAAA9A8A7A6A5A4A3A2A1A0'  /* 5
EQU FILLER     (16)  X'9F9E9D9C9B9A99989796959493929190'  /* 6
EQU FILLER     (16)  X'8F8E8D8C8B8A898887868584838281808'  /* 7
*
*                        0 1 2 3 4 5 6 7 8 9 A B C D E F      *
*
EQU FILLER     (16)  X'7F7E7D7C7B7A797877767574737271707'  /* 8
EQU FILLER     (16)  X'6F6E6D6C6B6A696867666564636261606'  /* 9
EQU FILLER     (16)  X'5F5E5D5C5B5A595857565554535251505'  /* A
EQU FILLER     (16)  X'4F4E4D4C4B4A494847464544434241404'  /* B
EQU FILLER     (16)  X'3F3E3D3C3B3A393837363534333231303'  /* C
EQU FILLER     (16)  X'2F2E2D2C2B2A292827262524232221202'  /* D
EQU FILLER     (16)  X'1F1E1D1C1B1A191817161514131211101'  /* E
EQU FILLER     (16)  X'0F0E0D0C0B0A090807060504030201000'  /* F
*
*                        0 1 2 3 4 5 6 7 8 9 A B C D E F      *
```

```
EQU TRNT-FLDA        (20)  C'BLUEJAYS EAT PEANUTS'
EQU TRNT-FLDB        (256)
EQU TRNT-FLDB
*                        0 1 2 3 4 5 6 7 8 9 A B C D E F
*
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 0
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 1
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 2
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 3
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 4
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 5
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 6
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 7
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 8
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* 9
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* A
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* B
EQU FILLER       (16)  X'00000000000010000000000000000000'  /* C
EQU FILLER       (16)  X'00010000000200000000000000000000'  /* D
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* E
EQU FILLER       (16)  X'00000000000000000000000000000000'  /* F

EQU IF-FLDA          (1)-P  P'-1'
EQU IF-FLDB          (2)-P  P'-1'
EQU IF-FLDC          (2)-P  P'1'
EQU IF-FLDD          (1)-P  P'1'
EQU IF-FLDE          (1)-P  P'1'
EQU IF-FLDF          (1)-P  P'1'
EQU IF-FLDG          (1)-P  P'1'
EQU IF-FLDH          (10)   C'BLUEJAYS'
EQU IF-FLDI          (4)    C'UEJA'
EQU IF-FLDJ          (4)-B  X'00000009'
EQU IF-ABCD          (4)    C'ABCD'
EQU IF-NAME          (4)    C'1234'

*********************************************************************
* ===>  SAMPLE OF "OR"      INSTRUCTION
*********************************************************************

    MOVE C'"OR  "   INSTRUCTION'  TO PRT1
    PRINT
    OR OR-FLDA WITH OR-FLDB
    PRINTHEX OR-FLDA
    PRINTHEX VAL223
*
*  END OF 'OR'      EXAMPLE
*
*********************************************************************
* ===>  SAMPLE OF "AND"     INSTRUCTION
*********************************************************************

    MOVE C'"AND "   INSTRUCTION'  TO PRT1
    PRINT
    AND AND-FLDA WITH AND-FLDB
    PRINTHEX AND-FLDA
    PRINTHEX VAL223
*
*  END OF 'AND'     EXAMPLE
*
*********************************************************************
* ===>  SAMPLE OF "XOR"     INSTRUCTION
*********************************************************************

    MOVE C'"XOR "   INSTRUCTION'  TO PRT1
```

```
    PRINT
    XOR XOR-FLDA WITH XOR-FLDB
    PRINTHEX XOR-FLDA
    PRINTHEX VAL223
*
*  END OF 'XOR'     EXAMPLE
*
************************************************************************
* ===>  SAMPLE OF "TRAN"    INSTRUCTION
************************************************************************

* * * * *
* SHOULD GET THIS:
*
*   WSTXX-XX
*         32132311B331B2332111
*         DCBAEE7DFAECF8AEABCD
*        01..05...10...15...20
*
* * * * *
    MOVE C'"TRAN"   INSTRUCTION'  TO PRT1
    PRINT
    MOVE C'SHOULD GET THIS:'        TO PRT1
    PRINT
    MOVE C'WSTXX-XX'                TO PRT17
    PRINT
    MOVE C'32132311B331B2332111'    TO PRT26
    PRINT
    MOVE C'DCBAEE7DFAECF8AEABCD'    TO PRT26
    PRINT
    MOVE C' 01..05...10...15...20'  TO PRT24
    PRINT
    MOVE C'GOT THESE RESULTS:'      TO PRT1
    PRINT

    TRAN TRAN-FLDA WITH TRAN-FLDB
    PRINTHEX TRAN-FLDA
************************************************************************
* ===>  SAMPLE OF "TRNT"    INSTRUCTION
************************************************************************

*
* SCANS UNTIL THE 'E' IN BLUEJAY IS FOUND AND STOPS TEST.

    MOVE C'"TRNT"   INSTRUCTION'  TO PRT1
    PRINT
    MOVE C'PTR1-1 SHOULD GET A X"C5" OR E'  TO PRT1
    PRINT
    MOVE C'VAL224-228 SHOULD GET X"0100000004" ' TO PRT1
    PRINT

    TRNT TRNT-FLDA WITH TRNT-FLDB
    PRINTHEX TRNT-FLDA
    PRINTHEX PTR1
    PRINTHEX VAL224-228
*
*  END OF 'TRNT'    EXAMPLE
*
************************************************************************
* ===>  SAMPLE OF "IF"      INSTRUCTION
*       WITH:
*            NESTED IF
*            IF WITH PARENTHESES
*            IF /ELSE/ENDIF
************************************************************************
```

```
    MOVE C'"IF: NESTED/PARENTHESES/IF/ELSE/ENDIF" '  TO PRT1
    PRINT
    IF IF-FLDA = IF-FLDB
         MOVE C'A = B'     TO PRT1
      IF ( IF-FLDC = IF-FLDD   AND            /* SHOULD GET
           IF-FLDD = IF-FLDE )                /* SHOULD GET
         MOVE C'C=D=E'         TO PRT10       /* SHOULD PRINT
      ELSE                                   /*  ELSE
         MOVE C'C NE D/E'      TO PRT10       /* SHOULD NOT GET
      ENDIF
* NOTE:  NOTHING FOLLOWING SHOULD EXECUTE!!!
    ELSE                                     /*  ELSE
*
       MOVE C'A NE B'          TO PRT30       /* SHOULD NOT GET
      IF IF-FLDC = IF-FLDD
         MOVE C'C = D'         TO PRT30       /* SHOULD NOT GET
      ELSE
         MOVE C'C NE D'        TO PRT30       /* SHOULD NOT GET
      ENDIF
    ENDIF
    PRINT
    IF IF-NAME = C'1234'
        MOVE C'CHET'       TO IF-NAME      /* SHOULD GET
    ELSE
        MOVE C'LEE '       TO IF-NAME      /* SHOULD NOT GET
    ENDIF
    PRINTHEX IF-NAME
    GOTO EOJ
9999END
```

Examples 38A and 38B

IF Statement with Test Under Mask Operands

```
    *-------------------------------------------------------*
    * EXAMPLE 38A:                                          *
    * NOTE:  RUNS ONLY UNDER 16.0+                          *
    *                                                       *
    * IFTMSK1:                                              *
    *        TEST OF "IF" VERB WITH VARIOUS TEST MASKS.     *
    *         TEST UNDER MASK BY SPECIFYING A FIELD SIZE    *
    *         OF ONE BYTE AND THE FOLLOWING RELATIONAL      *
    *         OPERATORS:                                    *
    *           TMO:  ONES                                  *
    *           TMZ:  ZEROS                                 *
    *           TMNZ: NOT ZERO                              *
    *           TMM:  MIXED                                 *
    *                                                       *
    *-------------------------------------------------------*

EQU FILLER     WST0             /* STARTING POINT
EQU FLDA       (2)
EQU FLDA    /* REDEFINES
EQU FLDB       (1)     X'A6'     /* 1010 0110
EQU FLDC       (1)     X'35'     /* 0011 0101
  TITLE 'IFTMSK1:  FIELD SIZE OF ONE BYTE'

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMO TEST1 TRUE  '   TO PRT30
    PRINT
    IF FLDB TMO X'80'
       MOVE C'TEST1 TRUE' TO PRT10      /* SHOULD GET THIS CONDITION
    ELSE
       MOVE C'TEST1 FALSE' TO PRT10
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMO TEST2 FALSE '   TO PRT30
    PRINT
    IF FLDC TMO X'80'
       MOVE C'TEST2 TRUE' TO PRT10
    ELSE
       MOVE C'TEST2 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMO TEST3 FALSE '   TO PRT30
    PRINT
    IF FLDC TMO X'A6'
       MOVE C'TEST3 TRUE' TO PRT10
    ELSE
       MOVE C'TEST3 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMO TEST4 TRUE  '   TO PRT30
    PRINT
    IF FLDB TMO X'A6'
       MOVE C'TEST4 TRUE' TO PRT10      /* SHOULD GET THIS CONDITION
    ELSE
```

```
   MOVE C'TEST4 FALSE' TO PRT10
ENDIF
PRINT

MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
MOVE C'TMZ TEST1 FALSE '   TO PRT30
PRINT
IF FLDB TMZ X'80'
   MOVE C'TEST1 TRUE' TO PRT10
ELSE
   MOVE C'TEST1 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
ENDIF
PRINT

MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
MOVE C'TMZ TEST2 TRUE  '   TO PRT30
PRINT
IF FLDC TMZ X'40'
   MOVE C'TEST2 TRUE' TO PRT10      /* SHOULD GET THIS CONDITION
ELSE
   MOVE C'TEST2 FALSE' TO PRT10
ENDIF
PRINT

MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
MOVE C'TMZ TEST3 FALSE '   TO PRT30
PRINT
IF FLDB TMZ X'A6'
   MOVE C'TEST3 TRUE' TO PRT10
ELSE
   MOVE C'TEST3 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
ENDIF
PRINT

MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
MOVE C'TMZ TEST4 FALSE '   TO PRT30
PRINT
IF FLDC TMZ X'A6'
   MOVE C'TEST4 TRUE' TO PRT10
ELSE
   MOVE C'TEST4 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
ENDIF
PRINT

MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
MOVE C'TMNZ TEST1 TRUE  '   TO PRT30
PRINT
IF FLDB TMNZ X'42'
   MOVE C'TEST1 TRUE' TO PRT10      /* SHOULD GET THIS CONDITION
ELSE
   MOVE C'TEST1 FALSE' TO PRT10
ENDIF
PRINT

MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
MOVE C'TMNZ TEST2 TRUE  '   TO PRT30
PRINT
IF FLDC TMNZ X'14'
   MOVE C'TEST2 TRUE' TO PRT10      /* SHOULD GET THIS CONDITION
ELSE
   MOVE C'TEST2 FALSE' TO PRT10
ENDIF
PRINT

MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
```

```
    MOVE C'TMNZ TEST3 TRUE  '   TO PRT30
    PRINT
    IF FLDB TMNZ X'A6'
       MOVE C'TEST3 TRUE' TO PRT10      /* SHOULD GET THIS CONDITION
    ELSE
       MOVE C'TEST3 FALSE' TO PRT10
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMNZ TEST4 FALSE '   TO PRT30
    PRINT
    IF FLDC TMNZ X'40'
       MOVE C'TEST4 TRUE' TO PRT10
    ELSE
       MOVE C'TEST4 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMM TEST1 FALSE '   TO PRT30
    PRINT
    IF FLDB TMM  X'40'
       MOVE C'TEST1 TRUE' TO PRT10
    ELSE
       MOVE C'TEST1 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMM TEST2 FALSE '   TO PRT30
    PRINT
    IF FLDC TMM  X'01'
       MOVE C'TEST2 TRUE' TO PRT10
    ELSE
       MOVE C'TEST2 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMM TEST3 FALSE '   TO PRT30
    PRINT
    IF FLDB TMM  X'A6'
       MOVE C'TEST3 TRUE' TO PRT10
    ELSE
       MOVE C'TEST3 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMM TEST4 TRUE  '   TO PRT30
    PRINT
    IF FLDC TMM  X'A6'
       MOVE C'TEST3 TRUE' TO PRT10      /* SHOULD GET THIS CONDITION
    ELSE
       MOVE C'TEST3 FALSE' TO PRT10
    ENDIF
    PRINT

    GOTO EOJ
9999END
```

```
    *-----------------------------------------------------------*
    * EXAMPLE 38B:                                              *
    * NOTE:  RUNS ONLY UNDER 16.0+                              *
    *                                                           *
    * IFTMSK2:                                                  *
    *         TEST UNDER MASK LOW BY SPECIFYING A FIELD         *
    *         SIZE OF 2 BYTES AND THE FOLLOWING RELATIONAL.     *
    *         OPERATORS:                                        *
    *           TMO:  ONES                                      *
    *           TMZ:  ZEROS                                     *
    *           TMNZ: NOT ZERO                                  *
    *           TMM:  MIXED AND LEFTMOST BIT IS ZERO            *
    *           TMP:  MIXED AND LEFTMOST BIT IS ONE             *
    *                                                           *
    *-----------------------------------------------------------*

EQU FILLER     WST0              /* STARTING POINT
EQU FLDA       (2)
EQU FLDA    /* REDEFINES
EQU FLDB       (1)     X'A6'     /* 1010 0110
EQU FLDC       (1)     X'35'     /* 0011 0101
  TITLE 'IFTMSK2:  FIELD SIZE OF TWO BYTES'

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMO TEST1 FALSE '   TO PRT30
    PRINT
    IF FLDA TMO X'4211'
       MOVE C'TEST1 TRUE' TO PRT10
    ELSE
       MOVE C'TEST1 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMO TEST2 TRUE  '   TO PRT30
    PRINT
    IF FLDA TMO X'2020'
       MOVE C'TEST2 TRUE' TO PRT10      /* SHOULD GET THIS CONDITION
    ELSE
       MOVE C'TEST2 FALSE' TO PRT10
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMO TEST3 TRUE  '   TO PRT30
    PRINT
    IF FLDA TMO X'A635'
       MOVE C'TEST3 TRUE' TO PRT10      /* SHOULD GET THIS CONDITION
    ELSE
       MOVE C'TEST3 FALSE' TO PRT10
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMZ TEST1 FALSE '   TO PRT30
    PRINT
    IF FLDA TMZ X'2020'
       MOVE C'TEST1 TRUE' TO PRT10
    ELSE
       MOVE C'TEST1 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMZ TEST2 TRUE  '   TO PRT30
```

```
PRINT
IF FLDA TMZ X'4040'
   MOVE C'TEST2 TRUE' TO PRT10      /* SHOULD GET THIS CONDITION
ELSE
   MOVE C'TEST2 FALSE' TO PRT10
ENDIF
PRINT

MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
MOVE C'TMZ TEST3 FALSE '   TO PRT30
PRINT
IF FLDA TMZ X'A635'
   MOVE C'TEST3 TRUE' TO PRT10
ELSE
   MOVE C'TEST3 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
ENDIF
PRINT

MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
MOVE C'TMNZ TEST1 TRUE  '   TO PRT30
PRINT
IF FLDA TMNZ X'4210'
   MOVE C'TEST1 TRUE' TO PRT10      /* SHOULD GET THIS CONDITION
ELSE
   MOVE C'TEST1 FALSE' TO PRT10
ENDIF
PRINT

MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
MOVE C'TMNZ TEST2 TRUE  '   TO PRT30
PRINT
IF FLDA TMNZ X'1414'
   MOVE C'TEST2 TRUE' TO PRT10      /* SHOULD GET THIS CONDITION
ELSE
   MOVE C'TEST2 FALSE' TO PRT10
ENDIF
PRINT

MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
MOVE C'TMNZ TEST3 TRUE  '   TO PRT30
PRINT
IF FLDA TMNZ X'A635'
   MOVE C'TEST3 TRUE' TO PRT10      /* SHOULD GET THIS CONDITION
ELSE
   MOVE C'TEST3 FALSE' TO PRT10
ENDIF
PRINT

MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
MOVE C'TMM TEST1 FALSE '   TO PRT30
PRINT
IF FLDA TMM  X'0605'
   MOVE C'TEST1 TRUE' TO PRT10
ELSE
   MOVE C'TEST1 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
ENDIF
PRINT

MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
MOVE C'TMM TEST2 FALSE '   TO PRT30
PRINT
IF FLDA TMM  X'A030'
   MOVE C'TEST2 TRUE' TO PRT10
ELSE
   MOVE C'TEST2 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
```

```
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMM TEST3 FALSE '   TO PRT30
    PRINT
    IF FLDA TMM  X'A635'
       MOVE C'TEST3 TRUE' TO PRT10
    ELSE
       MOVE C'TEST3 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMP TEST1 TRUE  '   TO PRT30
    PRINT
    IF FLDA TMP X'8040'
       MOVE C'TEST1 TRUE' TO PRT10      /* SHOULD GET THIS CONDITION
    ELSE
       MOVE C'TEST1 FALSE' TO PRT10
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMP TEST2 FALSE '   TO PRT30
    PRINT
    IF FLDA TMP X'1010'
       MOVE C'TEST2 TRUE' TO PRT10
    ELSE
       MOVE C'TEST2 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
    ENDIF
    PRINT

    MOVE C'===> SHOULD GET FOLLOWING:' TO PRT1
    MOVE C'TMP TEST3 FALSE '   TO PRT30
    PRINT
    IF FLDA TMP X'A635'
       MOVE C'TEST3 TRUE' TO PRT10
    ELSE
       MOVE C'TEST3 FALSE' TO PRT10     /* SHOULD GET THIS CONDITION
    ENDIF
    PRINT

    GOTO EOJ
9999END
```

Chapter

5

Troubleshooting and Memory Requirements

Troubleshooting and Memory Requirements

VISION:Report contains debugging facilities to help you troubleshoot a program. This section contains information about program checks and how to report a problem to Technical Support (see the section Contacting Computer Associates in Chapter 1 for more information).

See the section Examples in Chapter 4 for an example of trouble shooting a VISION:Report program.

Program Check Routine

VISION:Report causes a program check routine to be called and executed when a program check occurs under the following conditions:

- VSE or MVS system supports the program check option.
- VSE VISION:Report user has OPTION STXITPC=YES.
- MVS VISION:Report user has OPTION SPIE=YES.

The program check routine produces the following results:

```
PROGRAM CHECK AT ADDRESS NNNNNN OCCURRED, CAUSE WAS
```

The message above appears on the printer and the system console. The message is slightly different for MVS and VSE.

All active VISION:Report areas are hex-printed on the printer. For example: you are reading two files, using working storage, printing, punching, and writing three output files. The following areas are hex-printed.

INF	DET	WST
PRT	PUN	OFA
OFB	OFC	VAL

If you have a TRACE ALL or a TRACE LAST50 active for the entire program, the trace output assists you in identifying which statement caused the program check. Since all active areas are hex-printed, you have sufficient data to identify the cause of your program check to assist you in determining what to do in order to avoid it.

Warning: Since VISION:Report uses the VSE STXITPC and the MVS ESPIE macros to accomplish the previous, do not execute these macros in a subroutine that VISION:Report calls. You can avoid the problem in MVS with the option SUBSPIE=NO, but it can be costly in terms of execution time.

Reporting Problems

Prior to calling Technical Support for assistance, have the following information available by using the procedures indicated.

Within the VISION:Report program, ensure that you specify the following statements:

Options:

SPIE=NO	MVS only
STXITPC=NO	VSE only
LISTOPT=YES	
TRACECT=1	Use to obtain a trace line of sequence number executed. Omit this if the trace is very lengthy, and turn the trace on when you anticipate where the problem occurs.

Example

```
OPTION STXITPC=NO,LISTOPT=YES,TRACECT=1
```

Execute the TRACE LAST50 statement once in the mainline of the program. This resolves most problems. Use TRACE ALL in rare circumstances; it can produce a voluminous amount of paper. If you know where or when the problem occurs, place the TRACE ALL shortly before the problem area to reduce the amount of tracing required.

- Ensure that you list the entire job stream as it appears during execution, including all JCL, VISION:Report program, console messages, printouts, and dumps.

  If you are a VSE user with NOLOG as a standard option, add a / / OPTION LOG,PARTDUMP statement immediately after the / / JOB statement. All the JCL that follows shows up on SYSLST.

- If you use ABEND-AID, turn it off. It intercepts the program checks and destroys a lot of information needed to debug the programs.
- To turn off ABEND-AID for VSE, add the following JCL statement in your job stream, with an asterisk followed by a blank (*) starting in position 1:

  ```
  * AAPARM DIS
  ```

  To turn off ABEND-AID for MVS, add the following JCL statement in your job stream, starting in position 1:

  ```
  //ABNLIGNR      DD   DUMMY
  ```

- Other possibilities: If there are VSAM files involved, it is useful to have the IDCAMS DEFINE statements printed. If there are output files involved, run the IDCAMS DEFINE job immediately before the VISION:Report job. For other files, it is useful to dump records around the time that the abend occurs.

  Have the VISION:Report release number, the operating system release number, and any other pertinent information available.
- If MVS, check with your systems programmer to ensure that the dump produced contains all subpools, from 0 to 255 inclusive.

Memory Dumps

When a program check occurs, VISION:Report sets the ABEND or cancel code to 3336. When the VISION:Report OPTION U336DMP=YES is in effect (with SYSUDUMP DD on MVS), a memory dump of the partition is produced. No memory dump is printed when U336DMP=NO is in effect.

Storage Requirements

In general, VISION:Report runs in a partition or region of 256K-512K, depending on VISION:Report program size, table space, number of files, and I/O buffers, as well as any interfaces or called programs. If you use the VISION:Report Interface to DB2, in most cases a 1M partition or region is more than adequate.

STMTS, GENSIZE, LITSIZE, and #EQU

VISION:Report automatically attempts to dynamically obtain main storage in the partition or region if it is available. The STMTS, GENSIZE, LITSIZE, and #EQU option parameters, along with CALLCT and CALLSZ, are available for upward compatibility, but are no longer required. Previous releases of VISION:Report required that these parameters have specific limits and during the compilation stage. If you exceeded these limits, VISION:Report issued diagnostic messages and execution was not attempted.

Imperative statements generate one for one or more machine instructions.

Data and Table Space

VSE

The VSE user has the partition size less, approximately 64K for VISION:Report available for file I/O, file work areas, table loading space, and called subroutines. This availability can vary slightly between releases.

MVS

Use GETMAIN statements acquire I/O areas and table loading areas. The MVS user has the partition size less, approximately 64K for VISION:Report available for file I/O, file buffers, table loading space, and called subroutines.

File Sizes

MVS block size limit is 32,760 and record size is limited to 32,760 unless the format is variable spanned. VSE block size limit is 32,767. Record size limit for VSE is 32,767.

I/O Areas	No Buffering	Buffering
Fixed files - sum of block sizes	Times 1	Times BF
Variable Files – sum of block sizes	Times 1	Times BF
plus record sizes	Times 1	Times BF

BF=2 for VSE, BF=5 normally for MVS or installation standard or DD buffers override.

Chapter 6

Optional Material

Optional Material

Check with the system programmer who installed VISION:Report to ensure these materials were installed. For a description of the installation procedures for the MVS and VSE optional material, refer to the VISION:Report Installation Guide. Unless otherwise specified, the identifier or routine is applicable to both MVS and VSE.

Identifier	Routine or Program Function
DBOMPA (QJDBOMP) (VSE only)	DBOMP Interface
LIBR**** (VSE only)	CA-Librarian Assistance
QJCOBCVT	Convert COBOL copybooks to VISION:Report statements.
QJCOMREG (VSE only)	Subroutine to Access COMREG Area
QJEPRNT	VISION:Report to List Edit Masks
QJERAND	Random Number generator
QJJOBCOM (VSE only)	Subroutine to Access JOBCOM Area
QJPUNINT (VSE only)	3525 Punch/Interprets Subroutine
QUIKDATE, QUIKDATT	Date Calculation
QUIKDPRT	Print user date table
QUIKFLOP (VSE only)	3540 Floppy Disk Subroutine
QUIKIDMS (User must have a license)	CA-IDMS/DB Access Interface
QUIKILIB (MVS only)	CA-Librarian Interface Assistance
QUIKDLI (VSE only) QUIKIMS (MVS only) (User must have a license)	IMS-DL/I Interfaces

Identifier	Routine or Program Function
QUIKINCL (VSE only)	Source Statement Library Routine
QUIKIPAN (MVS only)	CA-Panvalet Subroutine
QUIKIPDS (MVS only)	PDS Include Subroutine
QUIKISAM (MVS only)	MVS ISAM Subroutine
QUIKISAM (VSE only)	VSE ISAM Macro
QUIKMOVE	Variable/Undefined Move Routine
QUIKPDS (MVS only)	PDS Routine
QUIKRPT	Multiple Reports Processor
QUIKTABL	Automated Tabling Routine
QUIKTIME	Time Subroutine
QUIKTRAN	ASCII/EBCDIC Translator
QUIKTRNT(MVS only) QUKBTRN (VSE only)	Translate Table
QUIKVEQU	EQU Statements for VAL Area
QUKBLIB (VSE only)	VSE Library Interface
TOTAL	TOTAL Interface
TOTAL4	TOTAL4 Interface

DBOMPA (QJDBOMP) — DBOMP Interface (VSE Only)

QJDBOMP allows you to access your DBOMP databases. A description of the calling sequences for Master File Processing, Chain File Processing, and Closing Files follows.

Call Format for Master File Processing

```
VERB  PHASENAME   OP1    OP2    OP3       OP4
CALL  QJDBOMP     AREA1  FILE   PROCESS   AREA2
```

Operands

- Address of the VISION:Report return area. Can be any unused I/O area if defined by the I/O parameter statement and the symbolic unit assigned to ignore (such as, INF).
- Name of the master file to be processed as defined by the DEFM1 or DEFM2 macro (such as, C'DID$PNM').
- Processing mode. The valid parameters are:

C'SEQ'	Sequential processing
C'GEN'	Generic (requires fourth parameter)
C'RAN'	Random (requires fourth parameter)
C'UPD'	Update.

- Address of your search argument which can be read from trigger statement or any other input (such as, INF9) or file.

Assume that the parameters for accessing the database have been generated into the access module.

Example 1

```
INFCARD
    MOVE ZEROES TO WST1-4-P              /* Initialize update CTR.
    GET                                  /* Read data statement.
020 CALL QJBDOMP SAV1 C'DID$PNM' C'SEQ'
    *                                    /* Read master file.
    IF SAV4-10 IS HIVALUE                       /* Test for end of file.
        GO TO 120.
    IF SAV5-7 NOT EQ C'945'              /* Is record to be updated?
        GO TO 020.
    MOVE INF1-6 TO SAV93-96-P            /* MOVE in effective date.
    MOVE ZEROES TO SAV97-100-P           /* Initialize field.
    CALL QJDBOMP SAV1 C'DID$PNM' C'UPD'
    *                                    /* Write updated record.
    ADD C'1' TO WST-4-P                  /* Count updates.
    GOTO 020
120 MOVE WST1-4-P TO WST18-31
    *                                    /* Display update count.
    DISPLAY WST11-31
    CALL QJDBOMP C'CLOSE'                       /* Force last record to be
written.
    GOTO EOJ
999 END
051575
/*
/&
```

Example 2

```
//  ASSGN SYS006,131
//  ASSGN SYS011,IGN
//  EXEC QUKBJOB
INFCARD
DETTAPE03000300SSYS011
    MOVE SPACES TO DET1-300              /* Blank return area.
    MOVE C'00' TO WST1-2                 /* Initialize limiting counter.
030 GET                                  /* Read statement input.
    DOHEADERS
    MOVE INF1-20 TO PRT1
    MOVE C'CARD INPUT' TO PRT25
    PRINT
040 IF INF1 EQ C'R'
        GOTO 640.                        /* Go to the random routine.
    IF INF1 EQ C'G'
        GOTO 830.                        /* Try the generic routine.
060 CALL QJDBOMP DET1 C'DID$PNM' C'SEQ'
    *       1     2        3        4
    *
    * 1 Module name called
    * 2 VISION:Report return area
    * 3 Master file to be referenced
    * 4 Processing mode
    *
    IF DET5-7 EQ X'FFFFFF'               /* Key field test for end of file.
        GOTO EOJ.
    PRINT                                /* Force line between groups.
    MOVE C'MASTER' TO PRT1               /* Format PRINT line.
    MOVE DET5-23 TO PRT1
    MOVE DET57-86 to PRT10
    MOVE DET55-56 TO PRT65
    MOVE DET87-94 TO PRT70
    PRINT
215 EXIT
    ADD C'1' TO WST1-2                   /* Count masters read.
    IF WST1-2 EQ C'25'                   /* Had enough?
        MOVE ZEROES TO WST1-2            /* Reset counter.
        GOTO 030.                        /*Yes- try for another trigger statement.
     IF DET24-27 EQ C'END.'              /* Any product structure records?
        GOTO 060.                        /* No- go get next master.
    IF DET44-47 EQ C'END.'               /* Any routing records?
        GOTO 480.                        /* No- continue with prod str chase.
    *
300 CALL QJDBOMP DET101 C'CHN' C'DID$PNM' C'DID$MRT'
    *       1     2        3        4         5
    *
    * 1 Module name called
    * 2 VISION:Report return area        Note: Print DET past mstr return area.
    * 3 Process a chain chase            This will hold mstr in your
    * 4 Name of master file              work area.
    *
    IF DET101-103 EQ C'EOC'              /* End of chain indicator.
        GOTO 480.
    MOVE C'RTG' TP PRT4
    MOVE DET106-109 TO PRT10             /* Seq number.
    MOVE DET132-193 TO PRT20             /* Cost, description & tool number.
    PRINT
    GOTO 300                             /* Continue routing chase.
    *
    *    Prod structure routine
    *
```

```
480 CALL QJDBOMP DET1 C'CHN' C'DID$PNM' C'DID$PST'
    *       1     2      3        4         5
    *
    * 1 Module name called
    * 2 VISION:Report return area        Note:  For product structure clause
    * 3 Process a chain file             component master is returned to
    * 4 Name of master file              addr specified and chain record
    * 5 Name of chain file               is returned at master length +1.
    *
    IF DET101-103 EQ 'EOC'               /* End of chain.
        GOTO 060.
    MOVE C'PROD STR' TO PRT4
    PERFORM 170 THRU 215                 /* Why code it twice?
    GOTO 480
    *
    *      Random retrieval
    *
640 CALL QJDBOMP DET1 C'DID$PNM' C'RAN' INF2
    *       1     2      3          4     5
    *
    * 1 Module name called
    * 2 Return area
    * 3 Master to be read
    * 4 Processing mode
    * 5 Address of user's random key (read from card this trip)
    *
    IF DET5 EQ C'*'                      /* "No hit" indicator.
       MOVE INF1-20 TO PRT1
       MOVE C'RANDOM READ NO HIT' TO PRT22
       PRINT
       GOTO 030.
    MOVE C'RANDOM' TO PRT4
    PERFORM 170 THRU 215
    GOTO 030
    *
    *   Generic processing routine
    *
830 CALL QJDBOMP DET1 C'DID$PNM' C'GEN' INF2   /* Same format as random.
    IF DET5-7 EQ X'FFFFFF'
        GOTO EOJ.                        /* Was it end of file?
    MOVE C'GENERIC' TO PRT4
    MOVE C'15' TO WST1-2                 /* Reset CTR for 10 more reads.
    GOTO 170
999 END
R89214
     1 IS BLANK FOR SEQ
R            047465*
/*
/&
```

The first example is a module to initialize a field to packed zeros. The SAV area is not needed for the application, so it was used as a DBOMP return area.

In the second example, the DET area was used as the VISION:Report return area. This area was generated by using a DET I/O statement for the size of the retrieved records and a // ASSGN SYSnnn, IGN where nnn is the symbolic unit number used in the I/O statement, columns 20-22.

The VISION:Report return area must be equal to the length of the largest master record plus the longest chain record to be retrieved.

The following indicators are returned to VISION:Report by the DBOMP module:

- End of File — high values (X'FF') are returned in the first six positions of the master file key area.
- End of Chain — EOC is returned in the first three positions of the chain file work area.
- No Record Found — asterisks (*) are returned in the first six positions of the master file key area.
- Generic read, no hit — the next record greater than the key furnished is returned.
- All other abnormal conditions force an abnormal termination.

Call Format for Chain File Processing

```
VERB  PHASENAME   OP1     OP2    OP3       OP4
CALL  QJDBOMP     AREA1   FILE   PROCESS   AREA2
```

Operands

- Address of user-defined return area (such as, DET101). Product structure retrievals return the component master at this address; the chain record is returned at this address plus the length of the master. For routing retrieval, the chain record is returned to this address.
- C'CHN' is the only valid keyword.
- Name of the master file (such as, C'DID$PNM').
- Name of chain file (such as, C'DID$MRT').

Call Format for Closing the Files

All file open processing occurs automatically as required. CLOSE is required only if you made an update to the file during processing. If an update was made, you must execute the following before EOJ, where QJDBOMP is the name of the module.

```
CALL QJDBOMP C'CLOSE'
```

LIBR**** — CA-Librarian Interface Assistance (VSE Only)

These programs are interfaces to CA-Librarian. The TLIBGET and DLIBGET interfaces are for CA-Librarian releases prior to 3.8 only. As part of the VISION:Report installation process, separate source and object modules exist for CA-Librarian releases 3.8 and higher and for prior releases. Be sure to consult the person who installed CA-Librarian to determine which release you are using, as well as consulting the CA-Librarian documentation for full details.

There are several differences between CA-Librarian release 3.8 and prior releases. Releases prior to 3.8 necessitated separate source (and object modules) for obtaining members for tape and disk. Starting with release 3.8, tape and disk accesses are incorporated into one module and allowed access to either tape or disk using specific sequences. The primary differences involve changing some fields in the source interface, which are documented in the Computer Associates source interface program. More detailed explanations can be found in the CA-Librarian Guide.

VISION:Report's release tape default is disk access first — if it cannot find a disk master, it will then look for a tape master. The default module name to specify in your CALL statement remains DLIBGET in order to maintain compatibility as much as possible. Determine what changes, if any, were made to the Computer Associates interface and if the module name to be called has been changed.

TLIBGET - Tape CA-Librarian GET & DLIBGET - Disk CA-Librarian GET

TLIBGET and DLIBGET are self-contained subroutines and can be entered by the CALL statement from VISION:Report. These routines process the CA-Librarian file (tape or disk) in module and/or ascending sequence, returning to the user program one data unit at a time.

```
VERB   PHASENAME     OP1     OP2     OP3

CALL   TLIBGET       AREA1   AREA2   AREA3
       DLIBGET
```

Operands

- An 8-byte area containing the selected module name or, if processing sequentially, blanks. This area must be left-aligned with trailing spaces. The module name or blank spaces must be placed into this area prior to all calls requesting a new module.

  For a tape master, modules can be retrieved in any order, but excessive tape positioning can result if the modules are not requested in ascending sequence.

For disk masters, modules must always be selected either modularly (module name specified) or sequentially (module name blank). If both modes of selection are used in the same program, the results are unpredictable.

- A 115-byte area into which the called routine places the following:

Bytes	Description
Bytes 1-80	Statement image
Bytes 81-88	Sequence number
Bytes 91-98	Date statement was added to master-MM/DD/YY
Bytes 101-108	Module name
Bytes 109-112	Module password
Bytes 113-115	Number of statements in module (packed decimal)

- When end of module is reached, this area is set to low values. When processing sequentially (requesting module is blank) and end of file reached or when processing modularly (requesting module is non-blank) and module cannot be found, this area is set to high values.
- Any non-blank character in this 1-byte area causes the called program to retrieve another module, whether sequentially or by module name. It is your responsibility to ensure the validity of this area.

When processing the tape master or backup file, you must include two assign statements in your VISION:Report JCL to unassign the cycle-file (if no checking for latest tape master is wanted) and to assign the tape file to a tape drive.

```
//  ASSGN SYSnnn,UA             Cycle file
//  ASSGN SYSnnn,CUU            Tape unit
```

QJCOBCVT — Convert COBOL copybooks to VISION:Report Statements

QJCOBCVT is a standalone program that enables a user to convert COBOL copybooks of data fields into VISION:Report EQU statements. This program is a separate utility program that a user would run outside of a VISION:Report program. After conversion, the ouput could then be placed into an Operating System's COPY library, or other librarian system such as CA-PANVALET, that can be subsequently accessed by a VISION:Report program.

There are certain COBOL statements that VISION:Report cannot translate. They include:

- COBOL Level 66
- COBOL Level 88
- OCCURS clauses
- Edit mask

In some cases, the location can be determined, but the corresponding logic cannot be executed. For example:

```
EQU OCCURS-FIELD1X WST689-693 /* OCCURS !!
EQU OUT-OF-BAL-IO-ACCUM-1 WST 1799-1816 /* EDIT MASK !!
```

In the above examples, these statements were appended with a /* at the end of the EQU statements, warning that an OCCURS clause or an edit mask may not be correct and will require examination and modification.

In the example below, asterisks (*) are placed in coloumn 1 for the Level 88 statements to indicate that they could not be converted, and some logic may have to be added.

```
*     88 POLICY-IS-D-W-OR-R    VALUE `D'`W'W `R'.
*     88 P21-STAT-GOOD         VALUE `G'.
```

Note: All statements marked with an asterisk (*) or a /* must be examined by the user to verify its correctness.

In all cases, an output "punched" file would be produced, and the user would have to examine the statements and it is the responsibility of the user to ensure that the conversion was successful.

Note: FD data names are not processed. Only data names in WORKING-STORAGE SECTION are processed, and are put in WST.

Prior to executing QJCOBCVT, you need to compile the COBOL program, and the output listing needs to be available on disk (record length 133 bytes, format FBA).

Note: The COBOL program must compile without errors, or return a code of 4 or less.

The option of MAP, whereby a Data Division Map is produced during COBOL compilation time, is required. The listing for the WORKING-STORAGE SECTION is also required.

The following is sample JCL for compiling a COBOL program.

MVS JCL

```
//COMPCOB1 JOB etc., COMPILE COBOL PROGRAM
//STEP1    EXEC  PGM=IGYCRCTL,REGION=2M,
//             PARM='ADV,MAP,TEST,RES,LIST'
//STEPLIB  DD  DSN=SYS1.COB2COMP,DISP=SHR
//SYSLIB   DD  DISP=SHR,DSN=CGL.COBCVRTR.COPYLIB
//SYSPRINT DD  DISP=SHR,DSN=CGL.COBCVRTR.LIST133(COBOL1)
//SYSLIN   DD  DSNAME=&&SYSLIN,UNIT=SYSDA,
//             DISP=(MOD,PASS),SPACE=(CYL,(1,1)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=2480)
//SYSPUNCH DD  DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE)
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN    DD  DISP=SHR,DSN=CGL.COBCVRTR.COPYLIB(COBOL1)
/*
//
```

VSE JCL

```
* $$ JOB JNM=COMPCOB1,LDEST=(CGL)
* $$ LST CLASS=A,LST=SYSLST,DEST=(,CGL)
* $$ PUN CLASS=A,PUN=SYSPCH,DEST=(,CGL)
// JOB COMPCOB1
// DLBL  QJPROC,'QJ.PROCLIB'
// EXTENT ,DOS004
// LIBDEF  PROC,SEARCH=QJPROC.PROC
// EXEC    PROC=QJTEST
// DLBL IJSYS06,'SYS006.TEMP',0
// EXTENT SYS006,DOS003,1,0,1,30
// ASSGN SYS006,DISK,VOL=DOS003,SHR
// DLBL IJSYS07,'SYS007.TEMP',0
// EXTENT SYS007,DOS003,1,0,1,30
// ASSGN SYS007,DISK,VOL=DOS003,SHR
// OPTION ERRS,SXREF,SYM,NODECK
// DLBL IJSYSLS,'CGL.COBCVRTR.LIST133',99/366   SYSLST ASSIGNED TO DISK
// EXTENT SYSLST,DOS006,1,0,1,30
  ASSGN SYSLST,DISK,VOL=DOS006,SHR
// EXEC IGYCRCTL,SIZE=IGYCRCTL,PARM='APOST,MAP'
    cobol source program
/*
/&
  CLOSE SYSLST,00E
* $$ EOJ
```

After compiling the COBOL program, you can then run QJCOBCVT. The following is sample JCL:

MVS JCL

```
//QJCOBCVT JOB etc.
//RUNIT    EXEC  PGM=QJCOBCVT
//STEPLIB  DD   DISP=SHR,DSN=QJ.PROD.LOADLIB
//SYSUDUMP DD   SYSOUT=*
//SYSPRINT DD   SYSOUT=*
//SYSPUNCH DD   DISP=SHR,DSN=CGL.COBCVRTR.PUNCH(COBOL1) 80-byte OUTPUT FILE
//INPUT    DD   DISP=SHR,DSN=CGL.COBCVRTR.LIST133(COBOL1)
//*  PRINT PUNCHED OUTPUT
//*  NOTE:  OPTIONAL STEP
//COPYIT   EXEC  PGM=IEBGENER,REGION=1M
//SYSPRINT  DD  SYSOUT=*
//SYSUT3    DD  UNIT=SYSDA,SPACE=(TRK,(1,1))
//SYSUT4    DD  UNIT=SYSDA,SPACE=(TRK,(1,1))
//SYSIN    DD  DUMMY
//SYSUT2   DD  SYSOUT=* DCB=(LRECL=125,BLKSIZE=3750,RECFM=VB)
//SYSUT1   DD  DISP=SHR,DSN=CGL.COBCVRTR.PUNCH(COBOL1)
//
```

VSE JCL

```
* $$ JOB JNM=QJCOBCVT,CLASS=A,LDEST=(,CGL)
* $$ LST CLASS=A,LST=SYSLST,DEST=(,CGL)
* $$ PUN CLASS=A,PUN=SYSPCH,DEST=(,CGL)
// JOB QJCOBCVT
// DLBL  INPUT,'CGL.COBCVRTR.LIST133',99/366
// EXTENT       ,DOS006
// DLBL  QJPROC,'QJ.PROCLIB'
// EXTENT ,DOS004
// LIBDEF  PROC,SEARCH=QJPROC.PROC
// EXEC    PROC=QJTEST
// EXEC QJCOBCVT,SIZE=512K
/*
/&
* $$ EOJ
```

After reviewing the punched output, make any necessary modifications and place in a COPY library as input to VISION:Report.

Optional Parameters

On the EXEC statement of the program, QJCOBCVT, there are two optional parameters that could be passed. These two optional parameters are:

Parameter 1: Storage name that user wishes to start at. This is a 3-byte field.

Examples: INF, OFA

Default: WST

Parameter 2: Increment value for offset of COBOL compiler listing output within Data Division. Some compiler default is 7.

Default: 24

Below is a partial sample output from a COBOL compile (note that each line does not show entire length):

```
1PP 5688-197 IBM COBOL for MVS and VM   1.2.0                    COB2
   LineID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-|
0/* COB2
   000006         000007  DATA DIVISION.
   000007         000008  WORKING-STORAGE SECTION.
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9
   000024         001800 01  MY-SAMPLE.
   000025         001900     05 MY-SAMPL-KEY.
   000026         002000      10 MY-SAMP-AUTONO.
   000027         002100       15 MY-SAMP-AUTO-INITIAL.
   000028         002200        20 MY-SAMP-AUTO-INIT-PRE            PIC X(003).  etc.
   000029         002300        20 MY-SAMP-AUTO-INIT-X              PIC X.
   000030         002400       15 MY-SAMP-AUTO-NUMBER               PIC X(006).
   000031         002500     05 MY-SAMP-OWNED-STATUS               PIC X(001).
   000032         002600          88 ASSIGNED-AUTO                           VALUE 'A'.
   000033         002700          88 FREERUNNER-AUTO                         VALUE 'F'.
   000034         002800          88 LEASED-AUTO                             VALUE 'L'.
   000035         002900          88 OWNED-AUTO                              VALUE 'O'.
   000036         003000     05 MY-SAMP-MLG-TAB.
   000037         003100      10 MY-SAMP-MILEAGE-RATE-GROUP OCCURS 12 TIMES INDEXED BY
   000038         003200         MY-SAMP-INDEX-1                   PIC S9(1)V9(4)  COMP-3.
```

Notice that the important data really starts in position 25 of the listing, or after the sequence number. This is the increment value for offset of COBOL compiler listing output.

Messages

There are messages that may appear in the QJCOBCVT output. Many of these are strictly warning messages, usually in conjunction with a statement that cannot be processed by QJCOBCVT. Thus, you may have to manually change whatever is required.

Below are the error messages, all prefixed with the constant “QJCOBCVT”, and a brief description of the diagnostic when further explanation is necessary:

ERR001 CANNOT HANDLE THIS CLAUSE/LEVEL NUMBER (OR INVALID)

This is either an invalid Level number, or more likely an 88 or similar level, which cannot be processed. You will have to manually make changes within a VISION:Report program if you wish to utilize this function in COBOL. If 88 or similar level, it will be made into a comment so you can still see the original statement.

ERR002 A "9" MUST FOLLOW AN "S" OR "V", EG: S9(5) OR V999

ERR003 NO LENGTH FOUND

There seems to be a length missing from the statement. Please examine and change if necessary.

ERR004 COMP-1 OR COMP-2 NOT SUPPORTED

ERR005 SYNC - NOT PROCESSED. SOME OUTPUT PRODUCED

QJCOBCVT does not handle this parameter. You may have to examine the input/output statement and make whatever changes are required.

ERR006 LENGTH NOT GIVEN YET OR INVALID. SOME OUTPUT PROCESSED

There seems to be a length missing, or invalid, from the statement. Please examine and change if necessary.

ERR007 INDEX. PROCESSED SOME

QJCOBCVT does not handle this parameter. You may have to examine the input/output statement and make whatever changes are required.

ERR008 DISPLAY-ST (ERLING) FOUND. PROCESSED SOME

QJCOBCVT does not handle this parameter. You may have to examine the input/output statement and make whatever changes are required.

ERR009 PROBLEM WITHIN LENGTH OF NUMBER

The length of the field is probably invalid. You may have to examine the input/output statement and make whatever changes are required.

ERR010 DATANAME LENGTH MAXIMUM EXCEEDED

There is a maximum of 34 for a dataname length. You may have to examine the input/output statement and make whatever changes are required.

ERR011 LEVEL NUMBER INCORRECT - SKIPPING TO NEXT STATEMENT

Somehow, the level number (possibly within or underneath another level number) is incorrect. You may have to examine the input/output statement and make whatever changes are required.

ERR012 BIG TABLE EXCEEDED - JOB ABORTED

QJCOBCVT issues a Storage request to the operating system to process all Data Division statements in memory. Presently, each storage request is for 1636 entries, each entry being 80 bytes. Six requests for storage are allowed, for a total of 9,816 entries; if a seventh request is required, the above message will appear.

There is a PCP available from Tech Support that will allow you to increase this amount.

ERR013 EDIT MASK CANNOT BE HANDLED - SKIPPING TO NEXT STMT

QJCOBCVT does not handle edit masks. You may have to examine the input/output statement and make whatever changes are required for VISION:Report.

QJCOMREG —Subroutine to Access COMREG Area (VSE Only)

QJCOMREG provides you with a convenient method of moving data from/to the VSE user partition communication region, allowing you to pass information between individual job steps.

```
VERB PHASENAME   OP1     OP2
CALL  QJCOMREG   FLDDEF  C'PUTCOM'
                          OR
                        C'GETCOM'
```

Operands

- Any valid VISION:Report area in the standard format of VISION:Report field definitions. This area is assumed to be 11 bytes long.
- C'PUTCOM' puts the data in flddef into the user communication region, replacing all 11 bytes.
- C'GETCOM' retrieves the data from the user communication region and places that information in flddef.

  If neither C'PUTCOM' or C'GETCOM' is specified, C'GETCOM' is the default.

The PUTCOM call does not automatically update @VAL-COMREG.

Examples

```
CALL QJCOMREG WST1 C'PUTCOM'

CALL QJCOMREG LAST-PROGRAM C'GETCOM'

CALL QJCOMREG LEN-LAST-RUN

CALL QJCOMREG INF21-31 C'PUTCOM'
```

QJEPRNT — List Edit Masks

QJEPRNT lists the edit mask table in a format suitable for reference. It is distributed as part of SAMPLIB.

The name of the module specified in the VISION:Report LOAD statement must be changed to the name of the user edit mask table. In the SAMPLIB, check the member, QJEPRNT, to ensure that the statement with sequence number 100 has QUIKEMSK if MVS, and QUKBEMSK if VSE. When executing the member QJEPRNT in MVS, add a SYSPRINT DD statement to your JCL.

QJERAND — Random Number Generator

QJERAND generates a positive random number. The routine provides a large range with a low reoccurrence factor. The number ranges between 0 and 2 to the 31st power, unless it is limited by placing a restriction in the calling area. The routine can be used under VSE and MVS; it is self-relocating, serially reusable, and less than 2K long.

QJERAND is automatically installed during the VISION:Report installation procedure.

When calling QJERAND, one data area pointer (consisting of two binary fields, each four-bytes long) is required. The routine returns the generated number in bytes 1 through 4 of this area. You can limit the range of the number by specifying a restriction in positions 5 through 8 of the calling area. Below is an example of a VISION:Report program invoking QJERAND, with an explanation.

```
OPTION LISTOPT=NO,STMTEND=80
    EQU PARMS         WST1-8                        /*Binary pointer
    EQU PARMS                                       /*Redefines parms
    EQU COUNTEM       WST101-104-P ZERO             /*How many to generate?
    EQU RANDNO-RETURN WST1-4-B                      /*Random no. generated
    EQU RANDNO-LIMIT  WST5-8-B                      /*Positions limit
    MOVE C'1234' TO RANDNO-LIMIT                    /*Limit to 4 positions
    MOVE C'RANDOM NUMBERS GENERATED' TO PRT60       /*Message text
    PRINT                                           /*Print the message
100 CALL QJERAND PARMS                              /*Call generator module
    IF COUNTEM EQ P'12'                             /*Stop at 12 numbers
          GOTO EOJ.                                 /*If counter = 12, stop
    ADD C'1' TO COUNTEM                             /*Increment the counter
    MOVE RANDNO-RETURN TO PRT69-72                  /*Move number to print
    PRINT                                           /*Print number generated
    GO TO 100                                       /*Generate next number
```

RANDNO-RETURN is the return area for the random number after the call. You should always check to ensure that the number returned is non-zero.

RANDNO-LIMIT is the area that limits the range of the generated number.

QJJOBCOM — Subroutine to Access JOBCOM Area (VSE Only)

QJJOBCOM provides you with a convenient method of moving data (up to 256 bytes) from/to a VSE partition's JOBCOM area, allowing you to communicate between jobs or job steps within a partition.

```
VERB  PHASENAME   OP1       OP2          OP3

CALL  QJJOBCOM    C'GET'    LENGTH       LOCATION
                    OR
                  C'PUT'
```

Operands

- A literal or VISION:Report area containing one of the following:
  - 'GET' to read JOBCOM data
  - 'PUT' to write JOBCOM data
- Length of JOBCOM data to read or write, in either of the following formats:
  - A 2-byte binary VISION:Report flddef (that is, WST1-2-B)
  - A 2-byte hexadecimal literal (that is, X'0050')

  The length must be in the range of 1 to 256 bytes decimal.
- The location of the JOBCOM data. You can use any valid VISION:Report Flddef that has sufficient length to contain the JOBCOM information.

Examples

```
CALL QJJOBCOM C'GET' X'0100' WST1

CALL QJJOBCOM C'GET' WST1-2-B WST11

CALL QJJOBCOM C'PUT' X'0064' WST1
```

Do not attempt to use this routine in a VISION:Report program running under ICCF. The JOBCOM macro is not supported by ICCF and will cause VISION:Report to have errors.

QJPUNINT — 3525 Punch/Interpret Subroutine (VSE Only)

QJPUNINT supports the model 3525 interpreting card punch with FUNC=I. When using this subroutine, the punched card's data length is assumed to be 80 bytes and is punched and interpreted on an 80/80 basis. QJPUNINT output is directed to SYS005.

```
VERB    PHASENAME   OP1
CALL    QJPUNINT    FLDDEF
```

Operands

Location of the data (such as, INF1, WST101).

To close the card file and ensure proper closing and interpreting of the last card, leave this operand blank.

Example

```
// JOB PUN-INT
// DLBL filename,'your.report.lib'
// EXTENT SYS001,vvvvvv
// LIBDEF *,SEARCH=(lib.sublib)
// ASSGN SYS005,CUU                        Assign 3525 device
// EXEC QUKBJOB
OPTION SEQCHK=NO
INFCARD
ATEND 200
100 GET                                    /* Read input record.
    MOVE INF1-80 TO PRT-1                         /* Move input to print file.
    PRINT                                  /* PRINT it.
    CALL QJPUNINT INF1                     /* Call program for punch/interpret.
    GOTO 100                               /* Go get next.
*
200 CALL QJPUNINT                          /* CALL to CLOSE punch file.
    GOTO EOJ                               /* Stop run.
    END
.
.   INF DATA
.
/*
```

QUIKDATE — Date Calculation

QUIKDATE performs various date calculations and conversions. This routine allows the year 2000 to follow the year 1999 by using a four-character length year field. It converts dates from one format to another (two-character year to four-character year), and adjusts a date forward or backward.

The routine consists of two parts: the actual processing program (QUIKDATE) and the user date table (QUIKDATT).

For programs requiring a four-digit year, you must change source programs and use the new version of QUIKDATE. No changes are required to currently operating programs.

QUIKDATT

The user default date table, QUIKDATT, is used by the processing program in many of the calculation and conversion routines. It has three portions: day status, control information, and holiday dates table. Maintenance of the holiday dates table is at your discretion. If you want to load a different dates table, see Function 12 —Load another date table.

Day Status

The day status associates each of the seven days of the week with a normal length of day value. Each day, numbered from 1 (Monday) to 7 (Sunday) has a length of whole (10), half (05), or off (00) associated with it.

Control Information

The control information defines the length of a normal work week, the weekend day, and default values for century (19 or 20) when a 2-character year is input for processing.

Holiday Dates Table

The holiday dates table is maintained usually by the person who installs your software products. It defines all days that are known to your organization as holidays, assigning a value that reflects the length of the holiday, either a full day (10) or a half day (05). It is possible to have more than one holiday dates table with different names.

Customizing QUIKDATT is discussed in the installation materials. An example is included to assist you in installing QUIKDATT.

QUIKDATE

The operands vary based on the function being performed. Each operand can be either a VISION:Report field definition (in character format only) or a character literal.

QUIKDATE provides the capability of specifying the first two characters of the new four-character year. This capability is in the form of an optional parameter for all functions using a two-character input date mask. This optional parameter (19, 20) is used as input for a QUIKDATE function (01-11). This function can produce output having a four-character year.

For all functions that have the optional parameter (for the century), the default value can be overridden by the optional Windowing Technique parameters in the QUIKDATT macro, as part of the QUIKDATE installation within VISION:Report. You should check with the person who installed VISION:Report at your site to see which Windowing Technique parameters, if any, were chosen.

```
CALL QUIKDATE FUNCTION OP1 OP2 OP3 OP4 OP5 OP6
```

Term	Description
Function	A two-character code indicating the function. A detailed list of each function and its operands is included later in this section.

Operands

- Usually, the date which QUIKDATE is to convert or base calculations. The MASK operand describes the format of the date. The input date must provide the entire date.
- If Julian, YY and DDD must be present.
- If Gregorian, MM, DD, and YY must be present.
- Short forms are not supported for input dates.
- This operand describes the format of the input date or converted date. The following masks are valid for QUIKDATE:

Each mask must be followed by one or more spaces.

MMDDYY	DDMMYY	YYMMDD	MMDDYYYY	YYYYDDD
DDMMYYYY	YYDDD	MM/DD/YY	MM/DD/YYYY	YYYYMMDD
DD/MM/YY	YY/MM/DD	DD/MM/YYYY	YYYY/MM/DD	MMYY

MM is the month
DD is the day
YY is the two-character year
YYYY is the four-character year
DDD is the Julian day of the year

Example

```
DATE=940131          MASK=C'YYMMDD  '

DATE=04/21/93        MASK=C'MM/DD/YY  '
DATE=19920410        MASK=C'YYYYMMDD '
```

- This operand, required only by certain functions, contains the result of a date calculation or conversion. The size of this area varies by function.
- In Function 08, this is the number of days you want to add to or subtract from the DATE. The adjust factor must be a five-character number if positive or six characters with a leading minus sign. For example, C'00031' adds 31 days to the DATE; C'-00031' subtracts 31 days from the date.
- An optional operand for Function 07. This one-character operand represents the day code (1 through 7) of the last day of the week. The default code is 7 (Sunday).
- An optional operand for producing a four-character year. If you do not specify this optional parameter, the value assigned is taken from the binary flags described in the first invocation of the HOLIDAY macro. The value specified should be either 19 or 20. All other functions are the same. However, there are now 11 more date masks available for processing four-character years. Functions 1 through 10 have one of these values: 4, 19, 20. Function 11 has one of two values: 19, 20.

Value	Description
4	Four-character year input format or prefix default value is taken from user dates table.
19	Two-character year prefix to expand an input date.
20	Two-character year prefix to expand an input date.

QUIKDATE Functions

This section details the QUIKDATE functions and each optional parameter.

Function 00 — Call QUIKDATE from a COBOL program

```
CALL 'QUIKDATE'  USING  FUNCTION  RETURN-CODE
```

Example in COBOL program

```
01  QD-FUNC   PIC XX            VALUE ZEROES.
01  QD-RC     PIC 9(4)          VALUE ZEROES   COMP.

CALL 'QUIKDATE'  USING  QD-FUNC   QD-RC.
```

- Only the Function and Return Code are required.
- Function code must be a 2-character field of '00'.
- Return Code is defined in COBOL as 9(4) COMP.

Note: If you are calling QUIKDATE from a COBOL program, this must be the first function to QUIKDATE.

Function 01 — Convert Julian date to Calendar date

```
CALL QUIKDATE FUNCTION   DATE     OUTPUT       RETURN       OPTIONAL
                                  MASK         AREA         PARAMETER
```

Note: Each mask must be followed by one or more spaces.

For all functions that have the optional parameter (for the century),you can override the default value with the optional Windowing Technique parameters in the QUIKDATT macro, as part of the QUIKDATE installation within VISION:Report. Check with the person who installed VISION:Report at your site to see which Windowing Technique parameters, if any, were chosen.

Example

```
CALL QUIKDATE C'01'     C'93060' C'MMDDYYYY ' WST1    C'19'
```

You can only omit the optional parameter; all others are required. If you omit the optional parameter, the Julian date is treated as being in the format YYDDD, and the prefix value for the year portion of the date is taken from the control information in the dates table. Definitions of the valid values are:

Value	Description
4	indicates a four-character Julian year input format.
19	the two-character year prefix to form a four-character date when input year is two characters.
20	the two-character year prefix to form a four-character date when input year is two characters.

If you input a four-character year date (optional parameter value is 4), and the output mask consists of a two-character year, then the leading two characters of the input year are truncated.

The returned value is a maximum of ten characters.

Function 02 — Convert Calendar date to Julian date

Extend Julian date to a four-character year. Specify the values, 19 or 20, only when the input mask is a two-character year.

```
CALL QUIKDATE FUNCTION      DATE     INPUT      RETURN     OPTIONAL
                                     MASK       AREA       PARAMETER
```

For all functions that have the optional parameter (for the century), you can override the default value with the optional Windowing Technique parameters in the QUIKDATT macro, as part of the QUIKDATE installation within VISION:Report. You should check with the person at your installation that installed VISION:Report to see which Windowing Technique parameters, if any, were chosen.

Example

```
CALL QUIKDATE C'02'    C'010194' C'MMDDYY '  PRT1     C'19'
```

You can only omit the optional parameter; all others are required. If you omit the optional parameter, the Julian date created has the two-character year format. Definitions of the valid values are:

Value	Description
4	Four-character Julian year output format. If the input mask indicates a four-digit year, that value is considered in formation of the date. If a two-digit year format is used, control information from the dates table determines the prefix.
Supply the following values only when the input mask is a two-character year:	
19	the two-character year prefix to form a four-character date when input year is two characters.
20	the two-character year prefix to form a four-character date when input year is two characters.

The returned value is a character string of five or seven characters.

Note: If calling QUIKDATE from COBOL, this must be the first CALL to QUIKDATE. This establishes where the return code is in your COBOL program, and is the equivalent of VAL46-49 in a VISION:Report program.

Function 03 — Compute number of days between two dates

```
CALL QUIKDATE FUNCTION   FIRST    FIRST    SECOND   SECOND   RETURN   OPTIONAL
                         DATE     MASK     DATE     MASK     AREA     PARAMETER
```

For all functions that have the optional parameter (for the century), you can override the default value with the optional Windowing Technique parameters in the QUIKDATT macro, as part of the QUIKDATE installation within VISION:Report. You should check with the person who installed VISION:Report at your site to see which Windowing Technique parameters, if any, were chosen.

Example

```
CALL QUIKDATE C'03'  '01134'  C'YYDDD ' '021331' C'YYMMDD ' WST15    C'20'
```

You can only omit the optional parameter; all others are required. If both input dates are four-character year formats, this parameter is not used and is ignored. For two-character year format masks or fixed year sizes, the values represent:

Mixed year sizes	
4	prefix for the shorter date will be taken from the control information of the dates table. This is the default if no optional parameter is specified.
19	this prefix will be assigned to the shorter date.
20	this prefix will be assigned to the shorter date.

Two-character years	
4	default values are taken from the dates table control information. This is the default if no optional parameter is specified.
19	indicates the prefix for the first date and assigns 20 as the prefix for the second date.
20	indicates the prefix for the first date and assigns 19 as the prefix for the second date.

The returned value is three packed bytes.

Function 04 — Convert a date to a day of the week

```
CALL     QUIKDATE     FUNCTION   DATE     MASK     RETURN     OPTIONAL
                                                    AREA       PARAMETER
```

Example

```
CALL QUIKDATE   C'04'   C'03/07/93'  C'MM/DD/YY ' WST12-21
```

You can only omit the optional parameter; all others are required. If the input date is in a four-character year format, this parameter is not used and is ignored. For two-character year format masks:

Value	Description
4	default values are taken from the dates table control information. This is the default if no optional parameter is specified.
19	this prefix will be assigned to the year.
20	this prefix will be assigned to the year.

The returned value is a maximum of ten characters. The first byte is the day (such as, 1=MON, 2=TUES) followed by the day's name spelled out (such as, MONDAY, TUESDAY).

Function 05 — Compute month ending date for a given date

```
CALL     QUIKDATE     FUNCTION   DATE     MASK     RETURN   OUTPUT     OPTIONAL
                                                    AREA       MASK     PARAMETER
```

For all functions that have the optional parameter (for the century), you can override the default value with the optional Windowing Technique parameters in the QUIKDATT macro, as part of the QUIKDATE installation within VISION:Report. You should check with the person who installed VISION:Report at your site to see which Windowing Technique parameters, if any, were chosen.

Example

```
CALL QUIKDATE C'05'    TRL-DATE C'YYMMDD ' PRT1 C'MM/DD/YYYY ' C'19'
```

You can only omit the optional parameter; all others are required. If the input mask contains a four-character year, this parameter is not used and is ignored. When input mask uses a two-digit year, the values are:

Value	Description
4	Default values are taken from the control information of the dates table. This is the default if no optional parameter is specified.
nn	Any two digits to indicate the century.

The returned value is a maximum of ten characters in the format of the output mask.

Function 06 — Compute number of days left in a calendar month

```
CALL     QUIKDATE      FUNCTION    DATE     MASK     RETURN
                                                    AREA
```

Example

```
CALL QUIKDATE C'06'   INF7     C'DDMMYY '    WST15
```

You do not use the optional parameter for this function. All others are required. The default century is taken from the date table.

The returned value is two packed bytes.

Function 07 — Compute week ending date for a given date

```
CALL    QUIKDATE    FUNCTION    DATE    INPUT    RETURN    OUTPUT    OPTIONAL    OPTIONAL
                                        MASK     AREA      MASK      WEEKEND     YEAR CODE
                                                                       CODE      PARAMETER
```

For all functions that have the optional parameter (for the century), you can override the default with the optional Windowing Technique parameters in the QUIKDATT macro, as part of the QUIKDATE installation within VISION:Report. You should check with the person who installed VISION:Report at your site to see which Windowing Technique parameters, if any, were chosen.

Example

```
CALL QUIKDATE C'07'  INF1    C'YYDDD ' PRT6  C'YY/MM/DD ' C'5'  C'20'
```

You can only omit the optional parameters; all others are required. There are two optional parameters for this function.

- The weekend code optional parameter allows you to override the dates table definition of which day ends the work week (the weekend day). A numeric value of 1 through 7, in character format, defines the last day of the work week (such as, 1 = Monday, 7 = Sunday).

- If you use the year optional parameter, the weekend code parameter must contain a value. If you do not want to override the weekend day defined on the dates table, code a dummy value of zero, character format (C'0').
- If you use the year end code optional parameter, you must use the weekend code optional parameter (see example). When used, if both masks contain the same number of year characters or the input mask is a four-character year, the year code optional parameter is not used and is ignored. When the output mask exceeds the input mask year characters, the values are:

Value	Description
0	If weekend code is not used.
4	Default values are taken from the control information of the dates table. This is the default if no optional parameter is specified.
nn	Any two digits to indicate the century.

The returned value is a maximum of 10 bytes in the format of the output mask.

Function 08 — Adjust a date up or down

```
CALL QUIKDATE    FUNCTION    DATE     INPUT   ADJUST   RETURN    OUTPUT   OPTIONAL
                                      MASK    FACTOR   AREA      MASK     PARAMETER
```

For all functions that have the optional parameter (for the century), you can override the default value with the optional Windowing Technique parameters in the QUIKDATT macro, as part of the QUIKDATE installation within VISION:Report. You should check with the person who installed VISION:Report at your site to see which Windowing Technique parameters, if any, were chosen.

Example

```
CALL QUIKDATE C'08' VAL71 C'YYDDD ' C'00031' PRT65 C'MM/DD/YY ' C'19'
```

You can only omit the optional parameter; all others are required. The optional parameter is ignored when the input mask is a four-character year. When it is a two-character year, the valid values are:

Value	Description
4	Default values are taken from the control information of the dates table. This is the default if no optional parameter is specified.
nn	Any two digits to indicate the century.

The returned value is a maximum of ten characters in the format of the output mask.

The adjust factor must be a five-character number if positive or six characters with a leading minus sign. For example, C'00031' adds 31 days to the DATE; C'-00031' subtracts 31 days from the date.

Function 09 — Compute a two-digit month code to alphabetic month

```
CALL QUIKDATE      FUNCTION      DATE     MASK     RETURN
                                                   AREA
```

Example

```
CALL QUIKDATE      C'09'   C'0488'     C'MMYY '       PRT1
```

You do not use the optional parameter for this function. All others are required.

The returned value is a character string of 9 bytes.

Function 10 — Compute number of work days between two dates

```
CALL QUIKDATE    FUNCTION   FIRST    FIRST    SECOND   SECOND   RETURN   OPTIONAL
                            DATE     MASK     DATE     MASK     AREA     PARAMETER
```

For all functions that have the optional parameter (for the century), you can override the default value with the optional Windowing Technique parameters in the QUIKDATT macro, as part of the QUIKDATE installation within VISION:Report. You should check with the person who installed VISION:Report at your site to see which Windowing Technique parameters, if any, were chosen.

Example

```
CALL QUIKDATE  C'10'   C'01001'  C'YYDDD ' C'001231'  C'YYMMDD ' WST15   C'20'
```

You can only omit the optional parameter; all others are required. If both input dates are four-character year formats, this parameter is not used and is ignored. For two-character format masks or mixed year sizes, the values are discussed separately:

Mixed year sizes	
4	prefix for the shorter date will be taken from the control information of the dates table. If you do not specify an optional parameter, this is the default.
19	this prefix will be assigned to the year.
20	this prefix will be assigned to the year.

Two-character years	
4	default values are taken from the dates table control information. If you do not specify an optional parameter, this is the default.
19	indicates the prefix for the first date and assigns 20 as the prefix for the second date.
20	indicates the prefix for the first date and assigns 19 as the prefix for the second date.

The returned value is 6 bytes packed and the value should be treated as containing one implied decimal digit.

If you assign a non-work day as a holiday, the results will be incorrect. For example, if Fridays are normally a half-work day and New Year Day falls on a Friday, indicate only a half-day off (work day values are 10=full day, 05=half day, 00=off day).

Function 11 — Checks for holiday and time off

```
CALL QUIKDATE      FUNCTION   DATE     MASK     RETURN      OPTIONAL
                                                AREA        PARAMETER
```

For all functions that have the optional parameter (for the century), the default value can be overridden by the optional Windowing Technique parameters in the QUIKDATT macro, as part of the QUIKDATE installation within VISION:Report. You should check with the person who installed VISION:Report at your site to see which Windowing Technique parameters, if any, were chosen.

Example

```
CALL QUIKDATE C'11'   C'00140'   C'YYDDD '   PRT80       C'20'
```

- You can only omit the optional parameter; all others are required.

- Dates of holidays in the user date table must be of the four-character year format. The optional parameter is used to determine the prefix for two-character input masks by the following values:

Value	Description
4	Default values are taken from the control information of the dates table. This is the default if no optional parameter is specified.
nn	Any two digits to indicate the century.

The returned value is two packed bytes indicating time off (00=no time off, 05=half day off, 10=full day off). The routine does not check to see if the date is a non-work day (weekend).

Function 12 —Load another date table

```
CALL QUIKDATE    FUNCTION    DATETABLE
```

Example

```
CALL QUIKDATE     C'12'        C'MYTAB    '
```

- The date table must be the name of the user date table as identified in the phase or load library. This must be an 8-byte character string, padded with blanks if necessary.
- The date table you specify is used until you specify another date table.
- If you invoke this function, the default date table name of QUIKDATT is used.

If the table name is invalid, you get a return code of 11 for any function that tries to use the table (even if it is only to look up the default century).

Function 13 — Provide date of the first working date of the month

```
CALL QUIKDATE    FUNCTION    DATEIN      INPUT    RETURN    OUTPUT    OPTIONAL
                                         MASK     AREA      MASK      PARAMETER
```

For all functions that have the optional parameter (for the century), you can override the default value with the optional Windowing Technique parameters in the QUIKDATT macro, as part of the QUIKDATE installation within VISION:Report. You should check with the person who installed VISION:Report at your site to see which Windowing Technique parameters, if any, were chosen.

Example

```
CALL QUIKDATE   C'13'  VAL71  C'YYDDD '  PRT65    C'MM/DD/YY '    C'19'
```

You can only omit the optional parameter; all others are required. The optional parameter is ignored when the input mask is a four-character year. When it is a two-character year, the valid values are:

Value	Description
4	Default values are taken from the control information of the dates table. If you do not specify an optional parameter, this is the default .
nn	Any two digits to indicate the century.

The returned value is a maximum of ten characters in the format of the output mask.

If there are no work dates in the month, a return code of 13 is set. The return code could be correct, depending upon QUIKDATT or the user date table.

Error Codes

A return code is placed in the VISION:Report area VAL46-49 after each call to QUIKDATE. If VAL46-49 is not 0000, an error occurred. The explanations for the return codes are described below.

```
0001Invalid function
0002Invalid number of parameters
0003Date code format invalid for function
0004Address in parameter list not valid
0005Invalid Julian date
0006Invalid calendar date input
0007Invalid year input
0008Invalid day code
0009Invalid date adjustment factor
0010Invalid month abbreviation
0011Invalid holiday table (user date table)
0012Invalid optional parameter value
0013No work days in the month.
```

You should check and then clear VAL 46-49 to avoid any possible conflicts in return codes, such as during EOJ.

The HOLIDAY macro constructs the dates in the HOLIDAY table. Refer to the VISION:Report Installation Guide.

Examples Using QUIKDATE

Example 1

Look through an Accounts Receivable file and print any invoices outstanding over 30 days.

```
SORT AREA RL80 ON CUST-NR INV#
CALL QUIKDATE C'08' VAL71-75 C'YYDDD '  C'-00031' WST1-5 C'YYDDD '
                                                        /* Subtract 31 days.
IF VAL46-49 IS NOT EQ TO ZERO                           /* Check for errors.
   PRINTHEX VAL46-49
   GOTO EOJ.
010 GET
    CALL QUIKDATE C'02' INF7-12 C'MMDDYY ' WST6-10      /* Convert invoice date
                                                        /* MMDDYY to Julian date.
IF VAL46-49
   PRINTHEX VAL46-49
   MOVE ZEROS TO VAL46-49
    IF BILL-AMT IS ZERO
        GOTO 010.
    IF WST6-10 IS GT WST1-5      /* Compare date in invoice to date 31 days ago.
       GOTO 010.                 /* Invoice date is current.
    MOVE COMPANY TO PRT1                 /* Invoice date outstanding, set up
print.
    MOVE BILL-AMT TO PRT40
    MOVE WST6-10 TO PRT60
    PRINT DOUBLE SPACED
    GOTO 010
999 END
```

Example 2

Compute how many days elapsed between the trial date and the date when the company signed the contract.

```
EQU COMPANY-NAME    INF1-36
EQU CONTRACT-DATE   INF222-227 D
EQU TRIAL-DATE      INF246-251 D
EQU NUMBER-DAYS     WST1-3-P
TITLE ' DAYS BETWEEN TRIAL DATE AND CONTRACT DATE'
REPORT COMPANY-NAME CONTRACT-DATE TRIAL-DATE NUMBER-DAYS
MOVE ZERO TO NUMBER-DAYS
010 GET
CALL QUIKDATE C'03' TRIAL-DATE C'MMDDYY 'CONTRACT-DATE C'YYDDD'
         NUMBER-DAYS
    IF VAL46-49 IS NOT ZERO
       MOVE ZEROS TO VAL46-49
       PRINTHEX VAL46-49
       GOTO EOJ.
    PRINT REPORT
    GOTO 010
999 END
```

QUIKDPRT — Print User Date Table

QUIKDPRT prints the contents of the user date table in a report format.

The user date table report is produced in two contiguous parts. The first part prints the weekday and miscellaneous sections of the table. This details the defined length of each day, the day designated as the end of the week, the normal work week length, and default prefixes for two-character year format date values.

The remainder of the report is a listing of the entries in the holiday dates portion of the table (vertically, in a five column display). The first page of this listing can contain up to 80 entries; subsequent pages can contain up to 125 entries. The entries on the last page appear in balanced or even-column length.

This program is designed to execute alone, in batch mode. STEPLIB (MVS) or LIBDEF (VSE) statements could be required to define the desired library if multiple tables exist. The output is directed to the SYSPRINT file in MVS and to the system logical unit SYSLST in VSE. The MVS SYSPRINT DD statement DCB parameter must include RECFM=M, as printer control characters are machine format. It must also define the record length as 133 characters, including control character. To route the output to a tape or disk device, provide the JCL statements to define the output data set. Execution is initiated by the statement:

```
//  EXEC PGM=QUIKDPRT
```

or

```
//  EXEC  PGM=QUIKDPRT,PARM='MYDATT'
```

Do not call QJBDTPRL using VISION:Report statements.

```
MYDATT                    H O L I D A Y   A N D   U S E R   D A T E   T A B L E   D I S P L A Y              DATE 10/25/95
                                        WEEKDAY AND MISCELLANEOUS ENTRIES                                        PAGE   1
         CODE    DAY      LENGTH
           1   MONDAY     1.0                                          WEEK END DAY  7 (SUNDAY)
           2   TUESDAY    1.0                                          WEEK LENGTH    5.5
           3   WEDNESDAY  1.0                                          4 CHARACTER YEAR PREFIX DEFAULTS:
           4   THURSDAY   1.0                                               INPUT   19
           5   FRIDAY     1.0                                               OUTPUT  19
           6   SATURDAY    .5
           7   SUNDAY      .0

                                                HOLIDAY ENTRIES
     DATE          TIME        DATE          TIME        DATE          TIME        DATE          TIME        DATE          TIME
JULIAN  CALENDAR   OFF    JULIAN  CALENDAR   OFF    JULIAN  CALENDAR   OFF    JULIAN  CALENDAR   OFF    JULIAN  CALENDAR   OFF

1900001 01/01/1900 1.0  1993151 05/31/1993 1.0  1994315 11/11/1994  1.0   1995359  12/25/1995  1.0   1998365  12/31/1998  .5
1900358 12/23/1900  .5  1993186 07/05/1993 1.0  1994328 11/24/1994  1.0   1995365  12/31/1995   .5   1999001  01/01/1999 1.0
1900359 12/24/1900 1.0  1993249 09/06/1993 1.0  1994358 12/24/1994   .5   1996001  01/01/1996  1.0   1999358  12/24/1999  .5
1992001 01/01/1992 1.0  1993284 10/11/1993 1.0  1994359 12/25/1994  1.0   1996359  12/24/1996   .5   1999359  12/25/1999 1.0
1992146 05/25/1992 1.0  1993315 11/11/1993 1.0  1994365 12/31/1994   .5   1996360  12/25/1996  1.0   1999365  12/31/1999  .5
1992185 07/03/1992  .5  1993329 11/25/1993 1.0  1995001 01/01/1995  1.0   1996366  12/31/1996   .5   2000001  01/01/2000 1.0
1992251 09/07/1992 1.0  1993358 12/24/1993  .5  1995149  05/29/1995 1.0   1997001  01/01/1997  1.0   2000359  12/24/2000  .5
1992286 10/12/1992 1.0  1993359 12/25/1993 1.0  1995185 07/04/1995  1.0   1997358  12/24/1997   .5   2000360  12/25/2000 1.0
1992316 11/11/1992 1.0  1993365 12/31/1993  .5  1995247 09/04/1995  1.0   1997359  12/25/1997  1.0   2000366  12/31/2000  .5
1992331 11/26/1992 1.0  1994001 01/01/1994 1.0  1995282 10/09/1995  1.0   1997365  12/31/1997   .5   2001001  01/01/2001 1.0
1992359 12/24/1992  .5  1994150 05/30/1994 1.0  1995315 11/11/1995  1.0   1998001  01/01/1998  1.0   2001358  12/24/2001  .5
1992360 12/25/1992 1.0  1994185 07/04/1994 1.0  1995327 11/23/1995  1.0   1998358  12/24/1998   .5   2001359  12/25/2001 1.0
1992366 12/31/1992  .5  1994248 09/05/1994 1.0  1995358 12/24/1995   .5   1998359  12/25/1998  1.0   2001365  12/31/2001  .5
1993001 01/01/1993 1.0  1994283 10/10/1994 1.0
```

QUIKFLOP — 3540 Floppy Disk Subroutine (VSE Only)

QUIKFLOP supports the model 3540 floppy disk unit, allowing you to read an input file, write an output file, and specify the logical record size. The 3540 floppy disk is not the same as a PC disk.

VERB	PHASENAME	OP1	OP2	OP3	OP4	OP5
CALL	QUIKFLOP	OPERATION	REC-AREA	FEEDBACK	RECSIZE	NO. DISKETTES INPUT

Operands

- Operation specifies the I/O action to take, valid entries are:
- C'READ' for floppy disk input on file QFLOPI.
- C'WRITE' for floppy disk output on file QFLOPO.
- C'CLOSE' for closing either one or both files (remaining operands not needed with close).
- The field definition of the starting position at which the data records are read into with the READ operand, or written from with the WRITE operand. The length of this REC-AREA is assumed to be equal to 80 or the RECSIZE specified. This area is filled with high values (X'FF') when end of file is reached on reading input records.
- The field definition of a one-byte area at which status feedback information is indicated. Possible values to expect are:

Blank	No error occurred.
E	EOF on input file.
1	Invalid recsize specified in Operand 4.
2	Invalid operand 1 specified.
3	Insufficient number of operands specified.

- The logical record size to use for reading or writing floppy disk records. The default of C'080' is used if not specified. When entered, it must be three EBCDIC numeric digits in the range of 001 to 128 bytes. This operand is optional.
- This operand allows multiple disks on input operations. Must be 2 bytes packed. OPER 4 (RECSIZE) must also be specified.

Call Examples

```
INPUT:   CALL QUIKFLOP C'READ' WST1 WST201 C'100'

OUTPUT:  CALL QUIKFLOP C'WRITE' INF1 WST1 C'128'

CLOSE:   CALL QUIKFLOP C'CLOSE'
```

VISION:Report Example

```
//  JOB FLOPPY
//  ASSGN SYS004,X'CUU'                          Input-Assign Device
//  DLBL QFLOPI,'INPUT',,DU                      -LABEL SET
//  EXTENT SYS004
//  ASSGN SYS005,CUU                             Output-Assign Device
//  DLBL QFLOPO,'OUTPUT',,DU                     -LABEL SET
//  EXTENT SYS005
//  EXEC QUKBJOB
100 CALL QUIKFLOP C'READ' WST1 WST201
    IF WST1-5 IS HIVALUE                         /* Chk for EOF.
        GOTO 300.                                /* Do CLOSE at EOF.
    IF WST201 IS NOT BLANK                       /* Ck feedback.
        PRINTHEX WST1-201                               /* Not blank, error.
        GOTO EOJ.                                /* Display area and stop run.
    *    User
    *       Processing
200 CALL QUIKFLOP C'WRITE' WST1 WST300 C'100'
    IF WST300 IS NOT BLANK                       /* Chk feedback.
        PRINTHEX WST1-300                               /* Not blank, error.
        GOTO EOJ.
    GOTO 100                                     /* Display area and stop run.
300 CALL QUIKFLOP C'CLOSE'                       /* Close diskette files.
    GOTO EOJ
999 END
/*
/&
```

The VISION:Report program shown above is for illustrating the use of QUIKFLOP in reading and writing disk files. Typically, most installations have only one disk unit which can either read an input file or write an output file. In this case, it would require two steps to copy disk to disk with an intermediate storage of the file.

QUIKIDMS — CA-IDMS/DB Access Interface(Optional feature)

QUIKIDMS is the VISION:Report interface to access all releases of CA-IDMS/DB Database files, including release 12.0. Requests to CA-IDMS/DB with VISION:Report are made by a CALL QJBIDMS. The integrated IDMS DML statements are not supported since QUIKIDMS is a load and go processor, eliminating the need for any preprocessing steps.

All CA-IDMS/DB functions are supported within the restrictions of the sub-schema being used. A complete list of DML functions and CALL formats can be found in the appropriate CA-IDMS/DB Guide.

QUIKIDMS provides you with access to the IDMS communications block in which the database furnishes information concerning the outcome of the requested service. You can reference this communications block in VISION:Report by the data area name PCB. This area contains the suggested initial values at the beginning of each execution of QUIKIDMS.

The communications block is 216 bytes long (PCB1-216). QUIKIDMS places the job name in PCB1-8 and initializes the error-status indicator to C'1400' at PCB9-12. The IDBMSCOM array located at PCB97-196 is used as a CALL argument, indicating the DML function. A copybook of the communications block is in the SAMPLIB, member name SSCTRL.

To use VISION:Report with CA-IDMS/DB, execute the program named QUIKIDMS in conjunction with the necessary JCL, followed by the VISION:Report statements. Make CA-IDMS/DB database requests with a CALL QJBIDMS including one to four parameters depending upon the particular request.

Example

A database service request in COBOL DML (that is, OBTAIN CALC record-name) is coded in one of the following ways in VISION:Report using the EQU and CALL verbs.

```
EQU OBTAIN-CALC PCB128                          /* Same as IDBMSCOM(32).
EQU OBTAIN-SUF PCB139                           /* Same as IDBMSCOM(43).
*
CALL QJBIDMS OBTAIN-CALC C'REC-NAME-LITERAL' OBTAIN-SUF
```

or

```
EQU COM-(32) PCB128                             /* Same as IDBMSCOM(32).
EQU COM-(43) PCB139                             /* Same as IDBMSCOM(43).
EQU RN       WST301-316                         /* Record name literal.
*
CALL QJBIDMS COM-(32) RN COM-(43)
```

CA-IDMS/DB database protocol for initialization requires you to perform the following DML functions:

- The first IDMS call request must be BIND run-unit designating the communications block area location (PCB1) and sub-schema name.
- For each record type in the database that is referenced, you must code a BIND record-name indicating the record-name literal and record location.
- An OPEN function is performed by READY usage-mode including the type of access to be made to the database.

Two examples of using VISION:Report with CA-IDMS/DB follow. They are shown to illustrate the recommended EQU (equate) statements for coding calls to CA-IDMS/DB. These examples are referencing the sample CA-IDMS/DB database (refer to the VISION:Report Installation Guide).

Example 1 shows the use of EQU statements which are abbreviated IDMS DML functions and calls to QJBIDMS to read the entire customer record base.

Example 2 shows the use of EQU statements which map to IDBMSCOM(nn) shown in the IDMS call formats. The calls to QJBIDMS return ORDOR records based upon the CALC key submitted in statements.

VSE JCL Example

```
// JOB QJIDMS1
// DLBL QJLIB,'your.VISION.lib'    Phase Library
// EXTENT   ,volser
// DLBL IDMS,'your.IDMS.lib'       Phase Library
// EXTENT ,volser
// LIBDEF   PHASE,SEARCH=(QJLIB.sublib,IDMS.sublib)
// ASSGN  SYS010,153
// DLBL   SYS010,'IDMS.SAMPLE.CUSTOMER',,DA
// EXTENT SYS010,volser
// ASSGN  SYS011,153
// DLBL   SYS011,'IDMS.SAMPLE.PRODUCT',,DA
// EXTENT SYS011,volser
// EXEC QUIKIDMS,SIZE=512K
  ...  VISION:Report statements as shown below
/*
/&
```

MVS JCL Example

```
//QJIDMS1  JOB  (800-0000,0000),'IDMS EXAMPLE'
//STEP1   EXEC  PGM=QUIKIDMS,REGION=512K
//STEPLIB  DD  DISP=SHR,DSN=your.VISION.loadlib
//         DD  DISP=SHR,DSN=your.IDMS.loadlib
//SYSPRINT DD  SYSOUT=*
//SYS010   DD  DISP=SHR,DSN=IDMS.SAMPLE.CUSTOMER
//SYS011   DD  DISP=SHR,DSN=IDMS.SAMPLE.PRODUCT
//SYSIN    DD  *
  ...  VISION:Report statements as shown below
/*
//
```

Example 1

```
*
OPTION SEQCHK=NO
*    'DML' Function EQUates
EQU BIND-RUN-UNIT   PCB155       /* BIND RUN-UNIT.
EQU BIND-RN         PCB144       /* BIND 'rec-name'.
EQU READY-RETRIEVE  PCB133       /* READY USAGE-MODE for retrieval.
EQU OBTN-1ST-RN-AN  PCB115       /* Obtain first 'rec-name' within 'area-name'
EQU OBTN-NXT-RN-AN  PCB107       /* Obtain next 'rec-name' within 'area-name'.
EQU FINISH          PCB98        /* Finish.
EQU OBTAIN-SUF      PCB139       /* Obtain suffix arg. (i.e., IDBMSCOM(43)).
EQU COMM-BLOCK      PCB1-216     /* IDMS communications block location.
EQU ERR-STAT        PCB9-12      /* IDMS error-status location.
EQU CUST-RN         SAV1-16      /* Customer rec-name literal.
EQU CUST-AN         SAV17-32     /* Customer record location (area).
EQU CUSTOMER        WST1-104     /* Customer record location.
    *
    HDR 1A 1 VISION:Report/ CA-IDMS/DB SAMPLE DATA BASE RETRIEVAL
    HDR 1B   $IPLDAT$   PAGE  $PG$
    *
10  MOVE SPACE TO SAV1-100       /* Blank SAV area for literal names.
    MOVE C'CUSTOMER' TO CUST-RN  /* Set-up rec-name literal.
    MOVE C'CUSTOMER-REGION' TO CUST-AN    /* Set-up area-name literal.
    *
20  CALL QJBIDMS BIND-RUN-UNIT COMM-BLOCK C'DEMOSSO3'  /* Bind run-unit.
25  CALL QJBIDMS BIND-RN CUST-RN CUSTOMER /* Bind rec-name.
30  CALL QJBIDMS READY-RETRIEVE  /* Ready usage-mode is retrieve.
35  PERFORM 900 THRU 990         /* Test error status.
    *
40  CALL QJBIDMS OBTN-1ST-RN-AN CUST-RN CUST-AN OBTAIN-SUF
                 /* Obtn 1st rn-an.
    GOTO 60
    *
50 CALL QJBIDMS OBTN-NXT-RN-AN CUST-RN CUST-AN OBTAIN-SUF
                 /* Obtn nxt rn-an.
60  IF ERR-STAT IS EQ TO C'0307'         /* Test status for EOF condition.
    GOTO 200.    /* Do close-up and EOJ.
    *
70  PERFORM 900 THRU 990         /* Test error status.
90  PRINTHEX CUSTOMER    /* PRINT customer record in hex format.
100 GOTO 50
    *
200 CALL QJBIDMS FINISH  /* Request finish.
    GOTO EOJ     /* Go to End-Of-Job.
    *
900 IF ERR-STAT IS EQ TO C'0000' /* Test for zero IDMS return codes.
    GOTO 990.    /* Equal, all is well.
    MOVE C'QUIKIDMS ABNORMAL STATUS' TO PRT1       /* Move err-msg to print.
    PRINT DOUBLESPACE    /* PRINT.
    PRINTHEX COMM-BLOCK  /* PRINT comm in hex.
    PRINTHEX CUSTOMER    /* PRINT cust record area.
    GOTO EOJ     /* Force End-Of-Job.
990 EXIT         /* Exit PERFORMed routine.
    END
```

Example 2

```
*
*    'DML' Function EQUates
EQU COM-(02)        PCB98              /* Finish.
EQU COM-(32)        PCB128             /* Find/obtain calc 'rec-name'.
EQU COM-(37)        PCB133             /* Ready usage-mode for retrieval.
EQU COM-(43)        PCB139             /* Obtain suffix arg. (i.e. IDBMSCOM(43)).
EQU COM-(48)        PCB144             /* BIND 'rec-name'.
EQU COM-(59)        PCB155             /* BIND run-unit.
EQU COMM-BLOCK      PCB1-216           /* IDMS communications block location.
EQU ERR-STAT        PCB9-12            /* IDMS error-status location.
EQU ORD-RN          SAV1-16            /* Ordor rec-name literal.
EQU ORDOR           WST1-40            /* Ordor record location.
EQU ORD-NUMBER      WST1-7             /* Ordor record (key).
*
INFCARD                                /* VSE only.
     HDR 1A 1 VISION:Report/ I D M S SAMPLE DATA BASE RETRIEVAL
     HDR 1B   $IPLDAT$   PAGE  $PG$
     ATEND 200                         /* At EOF, GOTO close routine.
     MOVE SPACES TO SAV1-100           /* Blank 'SAV' area for literal names.
     MOVE C'ORDOR' TO ORD-RN           /* Set-up rec-name literal.
     CALL QJBIDMS COM-(59) COMM-BLOCK C'DEMOSSO3' /* BIND run-unit.
     CALL QJBIDMS COM-(48) ORD-RN ORDOR /* BIND rec-name.
     CALL QJBIDMS COM-(37)             /* Ready usage-mode retrieve.
     PERFORM 900 THRU 990              /* Test error status.
     *
50   GET                               /* Read ordor trigger statements.
     MOVE INF1-7 TO ORD-NUMBER         /* MOVE ord-key number to ord location.
     *
60   CALL QJBIDMS COM-(32) ORD-RN COM-(43)     /* Obtain calc rec-name.
     *
     IF ERR-STAT IS EQ TO C'0326'      /* Test status for no-record-found.
         MOVE INF1-20 TO PRT1          /* MOVE ordor-key number to ord location.
        MOVE C'ORDOR KEY NOT FOUND' TO PRT31
         PRINT DOUBLESPACE             /* PRINT the line.
         GOTO 50.                      /* Go process next statement.
     PERFORM 900 THRU 990              /* Test error status.
     PRINTHEX ORDOR                    /* PRINT ordor record in hex format.
     GOTO 50                           /* Go process next statement.
     *
200  CALL QJBIDMS COM-(02)             /* Request finish.
     GOTO EOJ                          /* GOTO End-Of-Job.
     *
900  IF ERR-STAT IS EQ TO C'0000'      /* Test for zero IDMS return codes.
         GOTO 990.                     /* Equal, all is well.
     MOVE C'QUIKIDMS ABNORMAL STATUS' TO PRT1  /* MOVE err-msg to print.
     PRINT DOUBLESPACE                 /* PRINT.
     PRINTHEX COMM-BLOCK               /* PRINT comm area in hex.
     PRINTHEX ORDOR                    /* PRINT ordor record area.
     GOTO EOJ                          /* Force End-Of-Job.
990  EXIT                              /* EXIT PERFORMed routine.
     END
```

QUIKILIB — CA-Librarian Interface Assistance (MVS Only)

If the CA-Librarian Interface package is installed on your system, you can use the CA-Librarian FAIR (File Access Interface Routines) routines. QUIKILIB is an interface subroutine, which processes the CA-Librarian file (disk or tape) by module name, returning to the user program one data unit at a time. QUIKILIB retrieves a member using the -INC statement. Nesting of -INC statements is supported.

QUIKILIB can be used by VISION:Report during the compile phase to retrieve items such as a complete VISION:Report program, table statements, EQU statements, or it can be used as a callable subroutine at execution time.

When QUIKILIB is used as a called routine, you must supply the called program two parameters.

```
VERB    MODULE     OP1    OP2
CALL    QUIKILIB   AREA1 AREA2
```

Operands

- An 80-byte area used as both a request and return area. Initiate a request with a ' -INC xxxxxxxx' statement (xxxxxxxx is the name of the desired CA-Librarian module) at this location. QUIKILIB returns the data to this area. You cannot initiate another request until the end of the current module is detected. When the last module you want is retrieved, terminate the CA-Librarian interface by placing five bytes of high values (X'FF') in this area.

  For a tape master, you can retrieve modules in any order, but excessive tape positioning could result if you do not request the modules in ascending sequence.

- An area of 101 bytes where QUIKILIB returns messages that are associated with return codes 8, 12, and 16.

  A return code is placed in VAL46-49 following each call. These should be checked to determine subsequent processing. Return codes are in character form:

Code	Description
0000	Data successfully returned from CA-Librarian—call again.
0004	An -INC statement has successfully identified a module (no data returned)—call again.
0008	First parameter contains an error, second parameter contains a message.
0012	End of module has been detected; first parameter contains end message.

Code	Description
0016	Message from QUIKILIB - call again.
0020	CA-Librarian data set has been closed, do not call again.

QUIKDLI — DL/I Interface (VSE Only) (Optional)
QUIKIMS — IMS Interface (MVS Only) (Optional)

The terms IMS and DL/I are used interchangeably, unless specifically noted otherwise.

The program to execute or call is QUIKIMS for MVS and QUIKDLI for VSE. QUIKIMS is 100 percent compatible with QUIKDLI, so if you switch from MVS to VSE or vice-versa, your IMS/DL/I interface is the same. The QUIKIMS and QUIKDLI interfaces are IMS and DL/I-release independent.

The interface is implemented by a CALL QJBTDLI. The CALL format is exactly the same as you would code in COBOL or Assembler to communicate with IMS. You do not need to be trained to use IMS with VISION:Report. Each and every IMS rule and protocol is the same as in COBOL. All IMS functions are supported. You can insert, delete, and replace, as well as execute read-only calls.

Your current COBOL or Assembler-oriented PSB/PCBs are usable with no modifications required. You can make calls against any of the PCB numbers within a PSB. PL1 PSB/PCBs are not supported. You execute IMS or DL/I (as you currently do) and specify QUIKIMS or QUIKDLI as the application program to be executed.

VISION:Report allows 11 parameters in the QJ CALL, one of which states the name of the interface. Since IMS calls require a function parameter, a PCB parameter, and an I/O (work area) parameter at minimum, you have a maximum of seven optional Segment Search Argument (SSA) parameters available to you. This means qualified calls cannot exceed seven levels of IMS database segment depth.

A data area is available to VISION:Report to aid in coding database VISION:Report programs; this area is the PCB. PCB is a pointer only and not a data area for storing data. On entry to VISION:Report and before the first QJBTDLI call, PCB points to the first PCB in your PSB. Subsequent calls to IMS cause the PCB pointer to be modified to point to the PCB used in the last QJBTDLI call.

If you also have the VISION:Report Interface to DB2, be aware that it also uses the PCB parameter, although its usage is slightly different.

QUIKIMS and QUIKDLI Syntax

```
     PARM1     PARM2     PARM3      PARM4     PARM5     PARM6      PARM11
CALL QJBTDLI   C'FUNC'   C'PCBnn'   WORK/IO   SSA1      SSA2    ....SSA7
```

Note: The data pointed to by the QJ PCB is the actual PCB data within IMS. Make references in a read-only mode to the PCB; otherwise, vital data can be destroyed resulting in unpredictable results.

Parameters:

Parameters 1 through 4 are mandatory.

1. Name of interface. Must be QJBTDLI.
2. This is the function parameter. Must be a valid 4-digit IMS function code. This parameter can be the character literal of the function such as C'GNbb' or a VISION:Report area where the 4-digit function is found. See IMS or DL/I reference material for valid function codes.
3. This is the PCB parameter and can be in the range of 01 to 255. You should use a 2-digit numeric PSB number representing the PCB desired within your PSB.

   If the PSB has only one PCB and/or you want to use PCB number one, you should use the C'01'.

   This parameter can be the literal C'nn' of the appropriate PCB number or a VISION:Report area where a two-digit EBCDIC numeric PCB number is found.
4. This is the I/O work area into which segments are read or where you place/build a segment for insertion.

   This must be a VISION:Report area (such as, INF1, WST1, OFA1).

   Note: If a common I/O area is used for all calls, it must be as long as the longest segment.
5. through 11

   The SSA parameters 5 through 11 are optional to VISION:Report. IMS/DLI protocol must be observed as to whether SSAs are optional or required depending upon the function you are attempting to execute.

   This parameter can be in the form of a 9-byte character literal (of the segment name) if unqualified SSAs are desired, or it can be a VISION:Report area containing the formal SSA. If all 9-byte positions are not used, there must be trailing spaces.

Equates to Make QUIKIMS Coding Easier:

```
EQU PCB-DBNAME   PCB1-8
EQU PCB-LEVEL    PCB9-10
EQU PCB-STATUS   PCB11-12
EQU PCB-PROCOP   PCB13-16
EQU PCB-RESERVED PCB17-20
EQU PCB-SEGNAME  PCB21-28
EQU PCB-KFBLEN   PCB29-32
EQU PCB-SENSEG   PCB33-36
EQU PCB-KFB      PCB37
```

VSE QUIKDLI JCL Example

```
// JOB QJDLI
// DLBL QJPHASE,'your.VISION.lib'    Phase Library
// EXTENT   ,volser
// DLBL DLIPHSE,'your.DLI.lib'       DLI Phase Library
// EXTENT   ,volser
// LIBDEF   PHASE,SEARCH=(QJPHASE.sublib,DLIPHSE.sublib)
   ...Data base assignment(s)
   ...DLBL/EXTENTs normal to your DL/I execution
   ...Any VISION:Report assignments, TLBL, DLBL/EXTENTs required
   ...
// EXEC DLZRRC00,SIZE=nnnK
      DLI,QUIKDLI,PSBName
      CALL QJBTDLI .....
   ...  (VISION:Report statements and calls go here)
/*
   ... (Instream data to VISION:Report, if any, goes here)
/*
/&
```

MVS QUIKIMS JCL Example

```
//QJIMS   JOB  (800-0000,0000),'IMS JCL EXAMPLE'
//STEP1   EXEC  PGM=DFSRRC00,PARM=,QUIKIMS,PSBName'
//STEPLIB  DD  DISP=SHR,DSN=your.VISION.loadlib
//IMS      DD  DISP=SHR...
//IEFRDER  DD  ...
//DFSRESLB DD  ...   IMS RESLIB
//DFSVSAMP DD  ...   Buffer Parameters
....
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  ...
//SYSIN    DD  *
      CALL QJBTDLI ...
      ...  (VISION:Report statements and calls go here)
/*
//
```

CALL Examples

Retrieve all root segments. Write those having an on hand value greater than $2,000.00 to an output file.

```
    EQU STATUS-CODE  PCB11-12
    EQU IO-AREA      OFA1
    EQU ON-HAND      OFA56-59-P

100 CALL QJBTDLI C'GN  ' C'01' IO-AREA C'ROOTSEGM '
    IF STATUS-CODE IS EQ C'GB'                          /* End of db?
        GO TO EOJ.
    IF STATUS-CODE IS EQ C'  '                          /* Ok call?
        IF ON-HAND IS GT P'200000'                      /* Gt 2k?
           WRITE OFA                                    /* WRITE to file.
           GO TO 100.                                   /* Go read next.
    PRINTHEX PCB1-50                                    /* Hexprint PCB.
           GO TO 100                                    /* Go read next.
```

Read the entire database. Accept all segments for inspection and possible use.

```
EQU STATUS-CODE PCB11-12
EQU IO-AREA WST1
100  CALL QJBTDLI C'Gn  ' C'01' IO-AREA
     IF STATUS-CODE IS EQ C'GB'                         /* Test for end.
          GO TO EOJ.
     IF STATUS-CODE IS EQ C'  ' OR                      /* Normal status?
          IF STATUS-CODE IS EQ C'GA' OR                 /* Cross a boundary?
          IF STATUS-CODE IS EQ C'GK'                    /* Diff seg type?
          GO TO 300.
*    Error messages should be issued here
*    PCB should be hex printed
*    One might want to print I/O area and/or SSAs
     GO TO EOJ                                          /* Terminate processing.
300  PROCESS SEGMENTS IN ANY MANNER DESIRED
    GO TO 100                                           /* Go read next segment.
```

Read the database and select only those segments where the field MF010010 is equal to 90.

```
EQU PCB-NO       WST30-31
EQU PCB-DBNAME   PCB1-8
EQU PCB-LEVEL    PCB9-10
EQU PCB-STATUS   PCB11-12
EQU PCB-PROCOP   PCB13-16
EQU PCB-RESERVED PCB17-20
EQU PCB-SEGNAME  PCB21-28
EQU PCB-KFBLEN   PCB29-32
EQU PCB-SENSEG   PCB33-36
EQU PCB-KFB      PCB37-137
000 MOVE C'01' TO PCB-NO
010 MOVE C'MF010000(MF010010 =90)' TO WST1-22
020 CALL QJBTDLI C'GN  ' PCB-NO DET1 WST1
030 IF PCB-STATUS EQ C'GB'
040      GO TO EOJ.
050 IF PCB-STATUS EQ C'  '
060      GO TO 110.
070 PRINTHEX PCB1-50
090 MOVE C'2048' TO VAL67-70
100 ABEND
110 MOVE DET1-10 TO PRT2
120 MOVE DET15-44 TO PRT20
130 PRINT
140 MOVE DET45-74 TO PRT20
150 PRINT
160 MOVE DET75-104 TO PRT20
170 PRINT
180 MOVE DET105-134 TO PRT20
190 PRINT
200 GO TO 020.
999 END
```

QUIKINCL — Source Statement Library Routine (VSE Only)

QUIKINCL retrieves a book from a VSE source statement library. It can be used by VISION:Report during the compile phase to retrieve items such as a complete VISION:Report program, table cards, EQU statements, from a source statement library**.**

QUIKINCL will not run on releases prior to VSE.

To invoke QUIKINCL, you can use an OPTION statement at execution time or permanently change the OPTION table for the installation to use the QUIKINCL routine whenever VISION:Report is used.

The OPTION keyword is 'UEXIT1' (that is, OPTION UEXIT1=QUIKINCL). Whenever QUIKINCL is used, a DLBL, EXTENT, and LIBDEF statement must be included to point to the appropriate library containing the book (or books) to be included. If the DLBL and EXTENT statements are in the standard label area, they need not be coded.

QUIKINCL retrieves modules by the use of a ++INCLUDE statement. Up to six levels of nesting of ++INCLUDE statements are supported.

```
++INCLUDE book-name
```

The syntax of the control statement defining the name of the book to QUIKINCL is:

- ++INCLUDE must start in column 1 of the statement.
- Book-name is the name of the book to be included. The book name can be from 3 to 10 characters long. There must be at least one space between the ++INCLUDE and the book name. The format of the book name is X.YYYYYYYY, where X is sublib identifier and YYYYYYYY is the book within that sublib. The period (.) is required.

  If the book name cannot be found, an error code is passed back to VISION:Report which causes a 3333 ABEND after all VISION:Report statements have been analyzed. An error message is also printed on the VISION:Report statement listing.

Example

This is an example of how to add an include member to your QUIKJOB Library.

```
// DLBL  FOR SOURCE
// EXTENT ,volser
// LIBDEF  FOR SOURCE
// EXEC    LIBR,SIZE=512K
   ACCESS  S=QJSRC.QJ150
    CATALOG  ARFILEI.Q          REPLACE=YES
************************************************************************
*                                                                      *
*********  ACCOUNTS RECEIVABLE  FILE DESCRIPTION  INPUT ***************
*                                                                      *
* FOLLOWING IS THE FILE DESCRIPTION FOR A/R FILE, SEQ, VSAM:           *
*                                                                      *
* BLOCKSIZE=5280, LRECL=352, FIXED BLOCK (IF SEQUENTIAL)               *
*                                                                      *
************************************************************************
EQU  AR-RECORD          INF1-352          /* Entire record
EQU  AR-ACCOUNT         INF4-10           /* Account number
EQU  AR-TRANS-DATE      INF38-43       D  /* Transaction date
EQU     AR-TRAN-MM      INF38-39          /*    MM
EQU     AR-TRAN-DD      INF40-41          /*    DD
EQU     AR-TRAN-YY      INF42-43          /*    YY
EQU  AR-BILL-DATE       INF44-49       D  /* Billing date
EQU     AR-BILL-MM      INF44-45          /*    MM
EQU     AR-BILL-DD      INF46-47          /*    DD
EQU     AR-BILL-YY      INF48-49          /*    YY
EQU  AR-CUST-NAME       INF85-109         /* Customer name
EQU  AR-STREET          INF110-134        /* Customer street addr
EQU  AR-STATE-ZIP       INF135-159        /* Customer city, state, Zip
EQU     AR-CITY         INF135-152        /*    City
EQU     AR-STATE        INF153-154        /*    State
EQU     AR-ZIP          INF155-159        /*    Zip
EQU  AR-BALANCE         INF170-174-P  2C  /*
EQU  AR-ACCT-CODE       INF182-183        /*
EQU  AR-INSTL-BAL       INF191-196-P  2C  /* Amt left on installment
EQU  AR-INSTL-PAY       INF197-201-P  2C  /* Planned installment paymt
EQU  AR-BAL-PARTPAY     INF202-205-P  2C  /* Actual payment
EQU  AR-INT-PARTPAY     INF206-208-P  2C  /* Interest, part payment
EQU  AR-NR-PAY          INF209-210-P  2C  /*
EQU  AR-KEY             INF211-219        /* Key:
EQU  AR-KEY-ACCT-CD     INF211-212        /*      Acct-code:
EQU  AR-KEY-ACCOUNT     INF213-219        /*      Account
************************************************************************
*
************   END OF ACCOUNTS RECEIVABLE FILE DESCRIPTION  ***********
*                                                                      *
************************************************************************
```

The following VISION:Report program was written to create a report from the AR file. The standard set of EQU statements for the AR file are to be used, plus a standard I/O specifications for the AR file.

```
// DLBL  FOR SOURCE
// EXTENT ,volser
// LIBDEF  FOR SOURCE
// DLBL  ARFILEI,'ARFILE.VSAM',,VSAM
// EXEC   QUKBJOB,SIZE=512K
OPTION UEXIT1=QUIKINCL,SEQCHK=NO
```

The following statement causes the I/O specification statement for the AR file to be included. This is an excellent way to make coding the VISION:Report easier; plus, if the attributes of the file change, the VISION:Report programs using this book will automatically use the current file attributes.

```
++INCLUDE Q.ARFILEI                              /* INF I/O spec card
```

The following statement causes a standard set of EQU statements to be included for the AR file. This technique is another excellent way of reducing programming changes whenever data is added, changed, or deleted from a record description for a file.

```
++INCLUDE Q.ARFILEI                              /* AR file INF equates
REPORT          AR-CUST-NAME
        AR-ACCT-CODE
        AR-ACCOUNT
010  GET
     PRINT REPORT
     GOTO 010
9999 END
/*
```

Nested INCLUDES

QUIKINCL supports up to 6 levels of nested ++INCLUDE statements. This allows you to do such things as include an output AR file in addition to an input AR file in one program. Assume the AR file for input has already been added as shown in the previous example and you are adding the EQU statements and I/O statement for an output AR file.

```
// DLBL  FOR SOURCE
// EXTENT ,volser
// LIBDEF  FOR SOURCE
// EXEC    LIBR,SIZE=512K
   ACCESS  S=QJSRC.QJ
    CATALOG  ARFILEO.Q           REPLACE=YES
EQU  AR-RECORD-O        OFA1-352          /* Entire record
EQU  AR-ACCOUNT-O       OFA4-10           /* Account number
 .....
EQU  AR-KEY-O           OFA211-219        /* Key:
EQU  AR-KEY-ACCT-CD-O   OFA211-212        /*      Acct-code
EQU  AR-KEY-ACCOUNT-O   OFA213-219        /*      Account
/+
    CATALOG  ARFILEOO.Q               REPLACE=YES
OFAKSDS    0352          LBL=ARFILEO
/+
    CATALOG  ARFLALL.Q            REPLACE=YES
++INCLUDE  Q.ARFILEI
++INCLUDE  Q.ARFILEO
++INCLUDE  Q.ARFILEIE
++INCLUDE  Q.ARFILEOE
/+
/*
```

The following VISION:Report program was written to copy the AR file to another VSAM file, dropping all records with a zip code of 91361. The standard set of EQU statements for the input and output AR file are used, plus the standard I/O specifications for both AR files.

```
// DLBL  FOR SOURCE
// EXTENT ,volser
// LIBDEF  FOR SOURCE
// DLBL  ARFILEI,'ARFILE.VSAM',,VSAM
// DLBL  ARFILEO,'ARFILE.VSAM.OUT',,VSAM
// EXEC   QUKBJOB,SIZE=512K
OPTION UEXIT1=QUIKINCL,SEQCHK=NO
++INCLUDE Q.ARFLALL                              /* Bring in input & output files
010  GET
     IF AR-ZIP  =  C'91361'
         GOTO 010.
     MOVE AR-RECORD TO AR-RECORD-O
     WRITE OFA
     GOTO 010
9999 END
/*
```

QUIKIPAN — CA-Panvalet Subroutine (MVS Only)

QUIKIPAN retrieves a member from a CA-Panvalet library. QUIKIPAN is distributed in source form. It retrieves a member by the use of a ++INCLUDE statement. Up to 6 levels of nesting of ++INCLUDE statements are supported.

There are two methods for using this subroutine.

- To include VISION:Report source code and/or table data from CA-Panvalet at compile time, either temporarily or permanently change the UEXIT1 option.

```
OPTION UEXIT1=QUIKIPAN
```

- At the point where you want the source to appear in your VISION:Report, code the following statement, starting in position 1:

```
++INCLUDE membername
```

- To retrieve a module at execution time, make a standard call to QUIKIPAN.

```
CALL QUIKIPAN WST1-80 PRT1-132
```

Because this subroutine is set up as a user exit module, there are some restrictions for using it as a callable subroutine. These restrictions are:

There are two required parameters:

- The first parameter passed must be an 80-byte area. This area serves as the request and the return area. To request a member, initialize the area to spaces, then move C'++INCLUDE ???????? ' to this area, where ???????? is the member you are requesting. QUIKIPAN returns one record from the member on each call; you cannot request another member until EOF has been reached on the previously requested member.
- The second parameter must be at least 101 bytes long. This area contains messages returned from QUIKIPAN. These messages are not optional, so you are required to supply this area. Messages are returned with the following return codes: 0008, 0012, 0016.

After processing is complete, call QUIKIPAN with high values (X'FF') in the first parameter. QUIKIPAN closes the CA-Panvalet data set. This is mandatory if you want to process more than one member.

The syntax of the control statement defining the member name to QUIKIPAN is:

```
++INCLUDE must start in column 1 of the statement.

Member-name is the name of the member to include. The member name can be from 1
to 8 characters long. There must be at least one space between the  ++INCLUDE
and the member name. A name less than 8 characters must be filled with blanks.
```

If the member name cannot be found, an error code is passed back to VISION:Report. This causes a 3333 ABEND after all VISION:Report statements have been analyzed. An error message is also printed on the VISION:Report statement listing.

The return code can be examined in VAL46-49. Possible codes returned are:

Code	Description
0000	The first parameter did not contain ++INCLUDE in position 1.
0004	The first parameter contains a data record from the requested member. Continue calling QUIKIPAN without altering the request/return area in any way to receive more data records.
0008	The first parameter contains an error. The second parameter contains the error message.
0012	EOF on the requested member. The first parameter contains the END message.
0016	QUIKIPAN placed a message in the message area. Call QUIKIPAN again without altering the request/return area in any way.
0020	The member is closed and acquired areas are free. Do not call QUIKIPAN again.

A JCL statement defining the CA-Panvalet data set must have the ddname PANVALET:

```
//PANVALET  DD   DSN= . . .
```

QUIKIPDS —PDS and PDS/E Include Subroutine (MVS Only)

The terms PDS and PDS/E are used interchangeably, unless otherwise noted.

QUIKIPDS retrieves a member from a partitioned data set (PDS). QUIKIPDS retrieves members by the use of a ++INCLUDE statement. Up to 6 levels of nesting of ++INCLUDE statements are supported.

There are two methods for using this subroutine.

- To include VISION:Report source code and/or table data from a PDS at compile time, either temporarily or permanently change the UEXIT1 option.

```
OPTION UEXIT1=QUIKIPDS
```

- At the point where you want the source to appear in your VISION:Report, code the following statement, starting in position 1:

```
++INCLUDE membername
```

To retrieve a module at execution time, make a standard call to QUIKIPDS.

```
CALL QUIKIPDS WST1-80 PRT1-132
```

Input to QUIKIPDS is defined by a JCL statement with ddname QUIKIPDS. This can be a single PDS or a list of concatenated partitioned data sets:

```
//QUIKIPDS   DD   DSN=FIRST.LIBRARY,DISP=SHR
(//          DD   DSN=NEXT.LIBRARY,DISP=SHR)
(            .                              )
(            .                              )
(            .                              )
(//          DD   DSN=LAST.LIBRARY,DISP=SHR)
```

```
++INCLUDE member-name
```

The syntax of the control statement defining the member name to QUIKIPDS is:

```
++INCLUDE must start in column 1 of the statement.

Member-name is the name of the member to include. The member name can be from 1
to 8 characters long. There must be at least one space between the  ++INCLUDE
and the member name. A name less than 8 characters must be filled with blanks.
```

If the member name cannot be found, an error code is passed back to VISION:Report. This causes a 3333 ABEND after all VISION:Report statements have been analyzed. An error message is also printed on the VISION:Report statement listing.

QUIKIPDS Used as a User Exit At Compilation Time

The following INF file EQU statements are present in a PDS member called PAYEQUS.

```
EQU DEPT       INF1-4
EQU EMP#       INF5-8
EQU YTD-GROSS  INF9-18
EQU YTD-FICA   INF19-28
EQU YTD-TAX    INF29-38
EQU SEX        INF50
```

The following VISION:Report program was written to create a report of all employees in department 1020 making less than $29,000 per year. The standard set of EQU statements for the PAYROLL file are to be used.

```
//LISTPAY JOB acct#,pgrname
//STEP0010 EXEC PGM=QUIKJOB
//STEPLIB DD DSN=your.vision.loadlib,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSN=...           DD STATEMENT FOR PAYFILE
//QUIKIPDS DD DSN=MY.PDS,DISP=SHR     (1)
//SYSIN DD *
OPTION UEXIT1=QUIKIPDS,SEQCHK=NO
++INCLUDE PAYEQUS                    (2)
TITLE 'PAYROLL ANALYSIS FOR DEPT 1020'
REPORT DEPT EMP# YTD-GROSS
010 GET
    IF YTD-GROSS LT C'0002900000'
        PRINT REPORT.
    GO TO 010
999 END
```

1 This statement is required. It points to the PDS library to include from.

2 This statement causes a standard set of EQU statements to be included for the PAYROLL file. This technique is an excellent way of reducing programming changes whenever data is added, changed, or deleted from a record description for a file.

Nested INCLUDES

QUIKIPDS supports up to 6 levels of nested ++INCLUDE statements. For example, you could include member TABLE3 whenever member TABLE1 or TABLE2 was included.

Examples

Assume the following VISION:Report table statements are present in a PDS member called TABLE1.

```
010 DEPARTMENT 010
020 DEPARTMENT 020
++INCLUDE TABLE2
100 DEPARTMENT 100
```

Assume the following VISION:Report table statements are present in a PDS member called TABLE2.

```
030 DEPARTMENT 030
040 DEPARTMENT 040
++INCLUDE TABLE3
```

Assume the following VISION:Report table statements are present in a PDS member called TABLE3.

```
060 DEPARTMENT 060
070 DEPARTMENT 070
080 DEPARTMENT 080
```

If the following VISION:Report were written, TABLE2 and TABLE3 would be included.

```
OPTION UEXIT1=QUIKIPDS,SEQCHK=NO
TABLSPEC 0100 01 03   04 20
010 GET
    IF INF1-3 IS ONTABLE
        MOVE INF1-3 TO PRT1                          /* Dept code
        MOVE TBH4-23 TO PRT6                         /* Dept name
        MOVE INF4-10 TO PRT40 2C                     /* YTD dept sales
        PRINT.
    GO TO 010
999 END
++INCLUDE TABLE1
```

In the previous example, if the ++INCLUDE TABLE1 was changed to ++INCLUDE TABLE2, TABLE3 would still be included. On the other hand, ++INCLUDE TABLE3 would not include TABLE1 and TABLE2.

QUIKIPDS Used as a Callable Subroutine At Execution Time

You can use QUIKIPDS as a callable subroutine at execution time but, because it was designed as a user exit for VISION:Report, there are some restrictions.

There are two parameters; both are required.

- The first parameter must be 80 bytes. This area serves as the request and return area. To request a member, move '++INCLUDE ????????' to this area, where ???????? is the member you are requesting. You cannot request another member until the previously requested member has reached EOF, with the exception that a ++INCLUDE in the requested member is processed.
- The second parameter must be at least 101 bytes. This area contains messages from QUIKIPDS.

After processing is complete, call QUIKIPDS with high values (X'FF') in the first parameter. QUIKIPDS closes the PDS and frees any acquired storage areas.

The return code can be examined in VAL46-49. The possible return codes are:

Code	Description
0000	The first parameter did not contain ++INCLUDE in position 1.
0004	The first parameter contains a data record from the requested PDS member. Continue calling QUIKIPDS without altering the request/return area in any way to receive more data records.
0008	The first parameter contains an error. The second parameter contains the error message.
0012	EOF on the requested member. The first parameter contains the END message.
0016	QUIKIPDS has placed a message in the message area. You must call QUIKIPDS again without altering the request/return area in any way.
0020	The PDS has been closed and acquired areas have been freed. Do not call QUIKIPDS again.

Note:

- You should clear the message area any time the return code indicates that a message was placed there, as QUIKIPDS does not do any clearing.
- The message area is formatted for VISION:Report use. Some messages will be indented and some will not. Carriage control characters will appear with some of the messages.

Example

Following is an example of how to use QUIKIPDS at execution time to retrieve the records from a member in a PDS.

```
//QUIKIPDS JOB acct#,pgrname
//STEP0010 EXEC PGM=QUIKJOB
//STEPLIB DD DSN=your.vision.loadlib,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//QUIKIPDS DD DSNAME=....
//SYSIN    DD *
OPTION SEQCHK=NO,LIST=YES
EQU WORKINGSTORAGE WST
EQU REQUEST-RETURN (80)     BLANKS
EQU MESSAGE-AREA   (101)    BLANKS
***
001 TITLE 'P D S   LISTING FOR MEMBER INFTAB1'
002 REPORT REQUEST-RETURN (RECORD..IMAGE)
***
010 MOVE C'++INCLUDE TAB1 ' TO REQUEST-RETURN  /* Set up request.
020 CALL QUIKIPDS REQUEST-RETURN MESSAGE-AREA  /* Call PDS module.
***
030 IF VAL46-49 EQ C'0004'                     /* Request-return contains data.
040     PRINT REPORT                           /* From the member, PRINT report
050     GO TO 020.                             /* and keep calling for more data.
***
060 IF VAL46-49 EQ C'0012'                     /* End of member - end message is
070     PRINT REPORT                           /* In request-return, PRINT it and
080     GO TO 900.                             /* Then tell QUIKIPDS to close PDS.
***
090 IF VAL46-49 EQ C'0008'                     /* Request is in error -the reason
100     PRINT REPORT                           /* Is in the message area, PRINT it
110     MOVE MESSAGE-AREA TO PRT1              /* And the message and then tell
120     PRINT                                  /* QUIKIPDS to close the PDS.
130     GO TO 900.
***
140 IF VAL46-49 EQ C'0016'                     /* Message from QUIKIPDS, possibly
150     MOVE MESSAGE-AREA TO PRT1              /* A nested include, print the msg,
160     PRINT                                  /* Clear the area, and keep calling
170     MOVE SPACES TO MESSAGE-AREA            /* For more data,
180     GO TO 020.
190 ABEND                                      /* Something must be wrong - ABEND.
***
900 MOVE HIVALUES TO REQUEST-RETURN            /* Indicate processing completed.
910 CALL QUIKIPDS REQUEST-RETURN MESSAGE-AREA  /* CALL PDS module.
920 IF VAL46-49 EQ C'0020'                     /* PDS has been CLOSEd – terminate
930     GO TO EOJ.                             /* Processing.
***
940 ABEND                                      /* Something must be wrong - ABEND.
999 END
/*
```

QUIKISAM —MVS ISAM Subroutine

MVS QUIKISAM allows you to randomly retrieve, update, and/or add new records to an ISAM data set. QUIKISAM can handle up to 9 different ISAM data sets during one given execution of VISION:Report. QUIKISAM can be called from Assembler or COBOL programs.

Random Retrieval

You must specify an area for the retrieved records and a key (same length as key on the file) of the retrieved record. If the record is not found, the subroutine passes high values back into this area. A return code of 12 is also placed in the VISION:Report return code in the values area (VAL46-49).

If the ISAM data set contains variable length records, the record returned is in the form of RL00DDDDDDDDDD.....DDDD where RL equals the record length in binary, 00 equals binary zero, and DDDD equals the record data.

Update

To update a record, read the record without changing the key. Then, specify the beginning location of the record to be updated and a parameter specifying C'UPDATE'.

You can change the RL portion of the variable length records to either shorten or lengthen records which are to be updated.

The subroutine does not allow records with a record length greater than LRECL of the data set to be rewritten. A return code of 8 is placed in the VAL return area (VAL46-49) and the record is not written if this condition is requested.

Add a Record

To add a record, specify its location as one parameter and C'ADD' as another.

Duplicate records (records with duplicate keys) are not added to the data set. A return code of 12 is placed in the VAL return code (VAL46-49) if a duplicate add is attempted.

Addition of variable length records should be in the format of RL00DDDDDDDD as explained above.

A return code of 8 is placed in the return code (VAL 46-49) if an attempt to add a variable record is made with the record length greater than the LRECL of the data set.

QUIKISAM CALL Formats

```
CALL QUIKISAM OPERAND2 OPERAND3 OPERAND4 [OPERAND5]
```

Random Retrieval

```
CALL QUIKISAM OPERAND2 OPERAND3 OPERAND4 OPERAND5
```

Operands

- QUIKISAM is the name of the subroutine in the library.
- C'READ'. This is the function operand. This requests a random read.
- VISION:Report area to return or place the ISAM record into.
- VISION:Report area containing the key of the record to be retrieved.
- C'ISAM1', C'ISAM2',.....C'ISAM9'. This is the ddname of the data set from which you want the retrieval to be made. If a ddname is not specified the default is ISAM1.

A DD statement is required for each data set to be referenced by the subroutine.

Example

```
CALL QUIKISAM C'READ' WST101 INF1-5 C'ISAM1'
```

The previous statement attempts to read a record from the ISAM1 data set into working storage beginning at WST101. The key of the record to be retrieved is found at INF1.

The fact that INF1-5 is specified does not specify a 5-byte key. The key length is taken from the information in the DD statement and/or data set. INF1 specifies where position one of the key begins.

```
CALL QUIKISAM C'READ' WST101 INF1
```

The previous call yields the same results as the prior one. ISAM1 data set is used by default.

Update

```
CALL QUIKISAM OPERAND2 OPERAND3 OPERAND4
```

Operands

- QUIKISAM is the name of the subroutine in the library.
- C'UPDATE'. This is the function operand. This requests an update.

- Symbolic address of the record to be rewritten (updated).
- C'ISAM1', C'ISAM2'....C'ISAM9'. This is the ddname of the data set you want updated. Default is C'ISAM1'

Example

```
CALL QUIKISAM C'UPDATE' WST101 C'ISAM1'
```

This statement attempts to update a record in the ISAM1 data set. The record begins at WST101.

The record to be rewritten must be the most recent read from the subject data set.

Add or Insert

```
CALL QUIKISAM OPERAND2 OPERAND3 OPERAND4
```

Operands

- QUIKISAM is the name of the subroutine in the library.
- C'ADD'. This is the function operand. This specifies an add or insert function.
- A VISION:Report area and starting position where the record to be added or inserted. The key should not already exist in the file.

  If the RKP is zero and the file is unblocked, the area should be the length of one record in the file plus the length of the key. The area should be built in KKKKDDDDDDDDDD..... where K represents the key and D represents the data.
- C'ISAM1', C'ISAM2'....C'ISAM9'. This is the ddname where the record is added or inserted. If you do not specify a ddname, the default is C'ISAM1'.

Example

```
CALL QUIKISAM C'ADD' WST201 C'ISAM1'
```

This statement attempts to insert or add a record which begins at WST201. The data set to be inserted into is ISAM1.

A return code of 12 is placed in the VAL return code area (VAL 46-49) if you attempt to add a record that already exists in the file. The action is not performed.

Call to Close Files

You should always close the files at the conclusion of your processing and prior to transferring to EOJ.

```
CALL QUIKISAM C'CLOSE'
```

This form of the call informs QUIKISAM to close all of the files.

Special Options

The QUIKISAM routine has three modes of operation:

- Random retrieval only. This mode (RANDOM) is the least expensive mode in terms of memory used.
- Random retrieval plus update. This mode (UPDATE) is the next most expensive in terms of memory used.
- Random retrieval plus update plus ability to add or insert records into the data set(s). This mode (ALL) is the most expensive in terms of memory used and is the default mode.

One call must be made for each file to be customized. If you want to customize a data set, you must make an option call before any other call which references the data set.

```
CALL QUIKISAM C'OPTION' C'ALL'         C'filename'
CALL QUIKISAM C'OPTION' C'UPDATE'      C'filename'
CALL QUIKISAM C'OPTION' C'RANDOM'      C'filename'
```

Filename could be ISAM1,ISAM2.....ISAM9, and is not an optional parameter.

Size of Routine

The basic QUIKISAM routine is approximately 4K. Areas such as I/O areas and work areas are dynamic and add to the 4K basic requirement.

BISAM is used as an I/O management method. The ultimate amount of memory needed to support QUIKISAM varies. Storage requirements depend upon the requested options (or defaults) and whether or not BISAM is resident.

Return Codes

When using this routine with COBOL, the return codes are available in COBOL's special return code register. When using this routine with Assembler, the return code is available in Register 15.

Record Formats and Space Requirements

Variable length records are retrieved in the following format. You should also use this as a guide to insert new records.

```
RL00DRDRDRDR............DR

Where RL equals binary record length

Where 00 equals binary zero

Where DR equals the data record.
```

The key is found embedded within the record at the appropriate position. If RKP is 12, LRECL is 76, and KL is 4, the record returned is 76 maximum long; RL is in position 1 and 2; the key is in position 13-16.

```
RL00DDDDDDDDKKKKDDDDD...........D

1-4      5-12   13-16       17-76
```

Variable Length

```
1  LRECL  75
   Unblocked     Data area required to retrieve record
   KL      5     into or build record for add=75
   RKP     4


                  RL00    KEY      DATA

                  1-4     5-9      10-75
2  LRECL  75
   Blocked      Same as format above
   KL      5
   RKP     4
```

Fixed Length

Fixed length records with RKP other than 0 are returned in the following format. You should also use this as a guide to insert new records.

```
3  LRECL  75
   Unblocked     Data area length required to retrieve
   KL      5     record into or build record for add=80
   RKP     0


                  KEY         DATA

                  1-5         6—80
4  LRECL  80
   Blocked       Data area required to retrieve record
   KL      5     into or build record for add=80
   RKP     0
                  KEY         DATA
                  1-5         6—80
5  LRECL  80
   Unblocked     Data area required to retrieve record
   KL      4     into or build record for add=80
   RKP    76


                  DATA       KEY

                  1-76       77-80
6  LRECL  80
   Unblocked     Data area required to retrieve record
   KL      5     into or build record for add=80
   RKP     4
                DATA    KEY      DATA
                1-4     5-9      10-80
```

Exceptions and Exceptional Conditions

QUIKISAM aborts with an ABEND code U3999 on the console and with an explanatory message on the printer under the following conditions:

- Missing DD statement.
- Invalid function for the file.
- Invalid file name (not ISAM1 through ISAM9).
- Too many or too few operands on a CALL request.
- Attempt to rewrite a record with a key other than that of the record read.
- Close has been issued. No subsequent attempts at I/O are allowed.
- A second or subsequent option type call is being requested, a previous option has been issued or defaulted to, and I/O has been performed. You can make one option type request prior to any I/O; no subsequent ones are allowed.
- Option issued for open file.
- Option invalid second operand.
- Program (QUIKISAM) logic error—should not occur.
- Invalid update attempt.
- Invalid add attempt.
- Attempt to update record not read.

QUIKISAM — VSE ISAM Macro

Any subroutine generated by the QUIKISAM macro has the necessary code to support random retrieval, updating a retrieved record, and adding a new record.

Random Retrieval

```
CALL PHASENAME OPERAND2 OPERAND3
```

Operands

- PHASENAME is the name of the subroutine as link-edited in the Phase library.
- The location of the starting position for the retrieved record. WST1 as an operand causes a retrieved record to be placed into WST1-xxxx. You must allow sufficient room for the record.

  You might want to use a file name not in use by the program to gain space for these purposes. Specify an input/output parameter for a file such as OFC, set the block length and record length the same, and assign the file IGN with an ASSGN statement in VSE. The SAV area can also be used.
- The location of the starting position for the key of the desired record. The key must be of the proper length as described to the QUIKISAM macro and consistent with the key of the file in question.

Example

```
CALL QUIKRAN WST201 INF1-xxx
```

This example calls a QUIKISAM generated routine named QUIKRAN. The retrieved record is returned into WST201-xxx. The key to retrieve the record is located at INF1-xxx. If the record cannot be found, WST201-xxx is filled with high values (X'FF').

Update

```
CALL PHASENAME OP2 OP3
```

Operands

- PHASENAME is the name of the subroutine as link-edited in the Phase library.
- The location of the starting position for the updated record.

The record must have been randomly read by the QUIKISAM routine as the last or immediately previous action performed by the QUIKISAM routine; reads for other records from the ISAM file or additions to the ISAM file must not have been performed since the read for this record was performed.

The QUIKISAM routine checks the key of the record you want to update against the key of the record read previously by the QUIKISAM routine; if they do not agree, the program aborts with a console message and/or abend code. This action guarantees the integrity of your files and protects you from error.

- C'UPDATE'. This is the function operand. This requests an update action against a previously read record.

Example

```
CALL QUIKRAN WST201 C'UPDATE'
```

This CALL example informs the routine QUIKRAN that the record to be updated against the previously read record's key is for blocked records only. This keeps you from committing logical errors such as trying to update a record that has not been read for update. It also keeps you from changing the contents of the key in the record.

No such check can be made for unblocked records.

Addition or Insertion of a Record

```
CALL PHASENAME OP2 OP3 OP4
```

Operands

- PHASENAME is the name of the subroutine as link-edited in the Phase library.
- The location of the starting position of the added or inserted record. The key should not already exist in the file.

  If the file is blocked, this area should be the length of one record in the file and the key should be embedded within the record at the same relative position as that of the ISAM file in question.

  If the file is unblocked, this area should be the length of one record in the file plus the length of the key. The area should be built in KKKKDDDDDDDDDDDDD... fashion; where K represents a key and D represents the data.
- C'ADD'. This is the function operand. This requests an add or insert to the file.

- (Optional.) A VISION:Report area one-byte long where a warning code is returned if the record to be added is a duplicate. If this is omitted, duplicates are not added to the file and processing continues on a normal basis. An EBCDIC 'D' (or hex X'C4') is placed at this location if a duplicate add is attempted. Duplicates are not added whether this operand is used or omitted.

Example

```
CALL QUIKRAN WST201 C'ADD' WST500
```

This example calls a routine called QUIKRAN. The action requested is an ADD to the file. If the record in question is a duplicate, QUIKRAN places a D at WST500. The record is found at WST201-xxx.

Close the File When Adding Records

```
CALL PHASENAME OP2
```

Operands

- PHASENAME is the name of the subroutine as link edited in the Phase library.
- C'CLOSE' This is the function operand and it requests the ISAM file to be closed.

Your end of job routine should always include a CALL phasename C'CLOSE' regardless of the functions that have been performed on the file.

QUIKMOVE — Variable/Undefined Move Routine

```
                        OP1             OP2         OP3
CALL     QUIKMOVE       FROM-AREA       TO-AREA     LENGTH
                                                    (2 BYTE BINARY)
```

QUIKMOVE moves data from one location to another, for the length of a value specified by you.

QUIKMOVE provides you with a convenient method of moving data with variable or undefined lengths, based upon a supplied value.

Note: To accomplish the same results in a more straightforward manner, you can also use the MOVE variable length verb instead of QUIKMOVE.

Operands

- The location of the first byte of the sending data area.
- The location of the first byte of the receiving area.
- The location of the value, which is used to indicate the length of the data movement, in sending and receiving locations.

The length specified must be in a 2-byte binary format. The maximum value allowed for the length operand is 32,767.

Examples

```
CALL QUIKMOVE INF1 OFA1 WST1
```

Variable-length record move:

```
CALL QUIKMOVE INF1 OFA1 INF1              /* RL in INF1-2-B
```

Undefined-length record move:

```
MOVE VAL1-4-P TO WST1-2-B                 /* RL in VAL1-4-P
CALL QUIKMOVE INF1 OFA1 WST1
```

Data perpetuation move:

```
MOVE C'*' TO CTA1                         /* Fill CTA1-1501 with '*'
MOVE C'1500' TO WST1-2-B
CALL QUIKMOVE CTA1 CTA2 WST1
```

Instead of using a CALL to QUICKMOVE, you can also code the following natively:

```
EQU  MY-CON              WST1         C'*'
EQU  MY-WST              (100)
EQU  MY-LEN              (2)-B        X'0064'
 MOVE  MY-CON  TO MY-WST  MY-LEN            /*  FILL  MY-WST  WITH '*'
```

QUIKPDS — PDS and PDS/E Routine (MVS Only)

```
CALL QUIKPDS  FUNCTION   OP1   OP2
```

The terms PDS and PDS/E are used interchangeably, unless specifically noted otherwise.

QUIKPDS allows updating and sequential and random retrieval of a PDS. QUIKPDS provides a convenient method of reading a PDS directory and optionally retrieving one or more members from the PDS. Sequential and random retrieval of PDS members are supported. Updating of records is supported by the OPEN-UPD and PUT functions.

Opening a PDS

```
CALL QUIKPDS C'OPEN' C'ddname  '
```

This CALL is only required if the ddname is different than the default ddname, which is QUIKPDS.

The ddname parameter is optional and, if used, must be 8 bytes — padded with spaces if necessary. If the ddname parameter is omitted, the default ddname of QUIKPDS is used. If QUIKPDS is already opened, the OPEN call will be ignored. Be sure to close the data set if you want to change the ddname.

```
CALL  QUIKPDS  C'OPEN-UPD'  C'ddname  '
```

This call is required if you want to retrieve records for updating, by PUT function.

Closing a PDS

```
CALL QUIKPDS C'CLOSE'
```

Along with closing the PDS, this call also frees the buffer areas. You should always code this call before going to EOJ. If QUIKPDS is not opened, the CLOSE call is ignored. If you use the DELUPGM=YES OPTION without the CLOSE call, the VISION:Report program abends.

Reading a Directory Entry

```
CALL QUIKPDS C'READ-DIR' member-name generic-lth
```

The member-name parameter is required and must be 8-bytes long, padded with spaces if necessary. This call establishes the area within VISION:Report which contains the directory entry, and is required before any member processing is allowed.

The generic-lth parameter is optional and, if used, must be a one-byte EBCDIC area containing a numeric digit of 1 through 8. This parameter implies sequential retrieval and retrieves only directory entries whose positions 1 through n match positions 1 through n of the member-area (where n is the contents of the generic-lth parameter). Do not use this parameter if you are doing random retrieval.

QUIKPDS can be patched to return the entire directory entry (not just the name) and reposition (point) to a member that has just been read. The member-name parameter must be expanded to include the entire directory entry. See the person who installed QUIKPDS for more information.

Reading a Member

```
CALL QUIKPDS C'GET' record-area
```

The record-area parameter is required and must be the same length as the record length of the PDS. If the PDS has undefined record formats (RECFM=U), the record-area must be the same length as the block size. For variable length records (RECFM=V), the area must be the same length as the maximum record length, including the 4-byte RDW.

If you opened the PDS using the OPEN-UPD function, the GET function retrieves each record in update mode by the PUT function. However, it is not necessary to issue a PUT function on a record that has been read in update mode. QUIKPDS automatically detects if any records in a block have been updated, rewriting the block only if updates have occurred.

Updating a Member

```
CALL  QUIKPDS  C'PUT'  record-area
```

This function updates the current record in the directory block that was previously selected by the GET function. The functions OPEN-UPD and GET must have been previously selected.

Checking the Return Code

QUIKPDS communicates the outcome of the call through the return code area (VAL46-49). Return codes are as follows:

Code	Description
0000	Call completed successfully.
0004	Member requested was not found.
0008	End of directory.
0012	End of member.

Sequential Retrieval of Directory Entries

To sequentially retrieve directory entries from a PDS, initialize the member-area with spaces and issue a READ-DIR call. Each call returns the next entry until the end of the directory is indicated by a return code 8 in VAL46-49.

```
EQU MEMBER-AREA WST1-8 SPACES
010 CALL QUIKPDS C'READ-DIR' MEMBER-AREA
    IF VAL46-49 EQ C'0008'                    /* End of directory?
        GO TO 900.                            /* Yes, go to CLOSE the PDS.
    MOVE MEMBER-AREA TO PRT1
    PRINT
    GO TO 010.
```

Random Retrieval of Directory Entries

To retrieve a specific directory entry randomly, move the 8-byte name of the member to the member-area and issue a READ-DIR call. Upon return, check the return code in VAL46-49 to determine if the member exists.

```
EQU MEMBER-AREA WST1-8
    MOVE C'ASMFCL  ' TO MEMBER-AREA
    CALL QUIKPDS C'READ-DIR' MEMBER-AREA
    IF VAL46-49 EQ C'0004'                    /* Member not found?
       ABEND.
```

Sequential Retrieval of All Members

To retrieve the actual members in a PDS sequentially, initialize the member-area with spaces and issue a READ-DIR call. Then issue GET calls until either the end of the member is indicated by a return code of 12 in VAL46-49, or you no longer want to retrieve any more records from that member.

In either case, when you want to start retrieval from the next member, simply reissue the READ-DIR call. You must also check for the end of the directory, which is indicated by a return code of 8 in VAL46-49.

Note: Do not alter the contents of the member area as this indicates that random retrieval is desired.

The name of the member currently being processed is available in the member-area.

```
EQU MEMBER-AREA WST1-8   SPACES
EQU RECORD-AREA WST9-88
010 CALL QUIKPDS C'READ-DIR' MEMBER-AREA
    IF VAL46-49 EQ C'0008'                      /* End of directory?
        GO TO 900.                              /* Yes, go to CLOSE the PDS.
040 CALL QUIKPDS C'GET' RECORD-AREA
    IF VAL46-49 EQ C'0012'                      /* End of member?
        GO TO 010.                              /* Yes, go get next dir.
    MOVE RECORD-AREA TO PRT1
    MOVE MEMBER-AREA TO PRT81                   /* Move member name
    PRINT
    GO TO 040                                   * Go get another record.
900 CALL QUIKPDS C'CLOSE'
    GO TO EOJ
999 END
```

Generic Retrieval of Directory Entries

To retrieve directory entries generically, initialize the member-area with the data on which you want the generic match. Then issue READ-DIR calls with the generic-lth parameter containing the length of the data to use in the match.

In the following example, all the entries prefixed with the characters MKT are returned until the end of the directory is indicated by return code 8 in VAL46-49.

```
MEMBER-AREA WST1-8   C'MKT     '
EQU RECORD-AREA WST9-88
020 CALL QUIKPDS C'READ-DIR' MEMBER-AREA C'3'     /* Length of 3 in member name
    IF VAL46-49 EQ C'0008'                        /* End of directory?
        GO TO 900.                                /* Yes, go close the PDS.
040 CALL QUIKPDS C'GET' RECORD-AREA
    IF VAL46-49 EQ C'0012'                        /* End of the member?
        GO TO 020.                                /* Yes, get next mkt dir.
    MOVE RECORD-AREA TO PRT1
    PRINT
    GO TO 040
```

In the above example, after each successful call for READ-DIR, the member-area contains the name of the next member found.

Random Retrieval of a Member

To randomly retrieve a member, issue a READ-DIR call with that member's 8-byte name in the member-area. If a zero return code is returned, issue GET calls until the end of the member is indicated by return code 12 in VAL46-49, or you no longer want to retrieve any more records from that member.

```
EQU MEMBER-AREA WST1-8
EQU RECORD-AREA WST9-88
    MOVE C'ASMFCL  ' TO MEMBER-AREA
    CALL QUIKPDS C'READ-DIR' MEMBER-AREA
    IF VAL46-49 EQ C'0004'                          /* Member not found?
        ABEND.
050 CALL QUIKPDS C'GET' RECORD-AREA
    IF VAL46-49 EQ C'0012'                          /* End of the member?
        GO TO 900.                                  /* Yes, go CLOSE the PDS.
    MOVE RECORD-AREA TO PRT1
    MOVE MEMBER-AREA TO PRT81
    PRINT
    GO TO 050                                       /* Go get another record.
```

Updating a Member

To retrieve all members of a PDS, and update each member, open the PDS for updating (OPEN-UPD), establish an area (READ-DIR with member), then retrieve (GET) each record within each member. In the example below, certain positions are updated with a constant, then the record is rewritten (PUT). Checks are made for end of directory, as well as end of member conditions. At end of job, the PDS is closed.

```
   *-----------------------------------------------------*
    *                                                     *
    *  READ A PDS(/E) FILE, ALL MEMBERS, AND UPDATE       *
    *  THE FIRST 5 RECORDS WITHIN EACH MEMBER.            *
    *                                                     *
    *-----------------------------------------------------*

    EQU  MEMBER           WST1-8         SPACES
    EQU  CARD             WST9-109       SPACES
    EQU  CARDIN           WST110-209     SPACES
    EQU  COUNT            WST300-301-P
    REPORT CARD
    MOVE C'MEMBER:' TO HDA12
    CALL QUIKPDS C'OPEN-UPD' C'QUIKPDS2'       /* Open for update
010 CALL QUIKPDS C'READ-DIR' MEMBER            /* Establish area
    IF @VAL-RETURN-CD EQ C'0008'               /* End of directory?
       GO TO 900.                              /* .. Yes, goto EOJ
    MOVE ZERO TO COUNT
    MOVE MEMBER TO HDA21
    DOHEADERS
040 CALL QUIKPDS C'GET' CARDIN                 /* Get record in
    IF @VAL-RETURN-CD EQ C'0012'               /* End of member?
       GO TO 10.                               /* .. Yes, get next member
    IF COUNT EQ P'5'
       GO TO 10.                               /* Do only 5
    MOVE C'***I DID IT***' TO WST130           /* Update this field
    CALL QUIKPDS C'PUT' CARDIN                 /* Update record
    MOVE CARDIN TO CARD
    ADD P'1' TO COUNT
    PRINT REPORT
    GOTO 40  /* Get next record w/in mem
900 CALL QUIKPDS C'CLOSE'
    GOTO EOJ
9999END
```

Error Messages

The following is a list of error messages that can be issued with a U0004 ABEND when fatal errors occur:

Message	Description
***QUIKPDS - OPEN ERROR	PDS is currently opened for read only. You will need to close the PDS, and then reopen with the function OPEN-UPD.
***QUIKPDS - PUT INVALID UNLESS OPEN-UPD IS CODED	PDS is currently opened for read only. You will need to close the PDS, and then reopen with the function OPEN-UPD.

Message	Description
***QUIKPDS - MEMBER AREA BLANK WITH RANDOM SEARCH	This message occurs only if you have modified the section of the QUIKPDS program by the label READEQUL, which is documented in the source program. The original QUIKPDS does not allow random positioning on a member twice in succession.
***QUIKPDS - NO MEMBER-AREA ESTABLISHED	A member-area must be established with a READ-DIR call before a GET call can be attempted.
***QUIKPDS - INVALID PARAMETERS DETECTED	The parameter list received is invalid. Refer to the section on calls and their operands.
***QUIKPDS - TRYING TO READ PAST END	An attempt was made to read past the end of the directory or the end of a member.

Note: If you do not consider the following recommendations, QUIKPDS can violate IBM standards and unpredictable results can occur.

- Read all the records of a member until you get the C'0012' end of member return code.
- For random retrieval, read the entire member.

QUIKRPT — Multiple Reports Processor

QUIKRPT provides extended reporting capabilities. VISION:Report supports one user report file on SYSPRINT or SYSLIST execution. QUIKRPT can support up to seven additional report files concurrently in the same execution of VISION:Report. It is a callable subroutine that provides support for commonly needed report features similar to VISION:Report. Check with the system programmer who installed VISION:Report to determine what customizations were made.

Using QUIKRPT, you can specify the following kind of declaratives: Report Headers and Linecount value. This is usually done one time in the housekeeping portion of the program. Other options include writing of print lines, page overflow, and printing of headers, as well as doing a CLOSE at end of job.

You pass report information to QUIKRPT by coding calls which include:

- The function to be performed.
- The values associated with the function.
- The report number to which the call applies.

The use of the ACCUM and BREAK, REPORT, PRINT REPORT, and other statements associated only with the REPORT verb are restricted in conjunction with any of the QUIKRPT files.

QUIKRPT Call Formats

In each of the QUIKRPT CALL formats noted, the last parameter shown (in parentheses) indicates the report number desired. The parameter is optional and, if omitted, defaults to RPT1. The only valid entries for the report number parameter are RPT1 through RPT7.

Declarative Functions

HDR Function — Header Locations

```
CALL QUIKRPT C'HDR' hdr1 hdr2 hdr3.....hdr8 (C'RPTn')
```

The HDR function defines the number of report headers and their respective locations. Up to eight heading lines can be specified for each report file used. Enter the VISION:Report field definition or beginning address of each heading area, in printing sequence, in the CALL statement. If none is entered, two blank lines are printed at the top of the form. The optional file name (default is RPT1) is the last parameter of the CALL statement.

Each header line is assumed to be at least 133 positions long, with the first position being an ASA carriage control character. Your data is assumed to occur in positions 2 through 133 of each line. Any valid ASA control character code can be used in the first position. The most common ones are listed below:

blank	Single-space before printing line.
0	Double-space before printing line.
-	Triple-space before printing line.
+	Suppress spacing before printing line.
1	Skip to channel 1 before printing.

QUIKRPT stores and tracks of the header address locations in memory. The header information must be maintained throughout the use of the report. You can modify the heading by performing another CALL HDR to the same RPTn number; this refreshes the number and locations of header line data.

When you change the value in a header line, you do not have to make another CALL QUIKRPT C'HDR'.

- There is no limit on the number of times QUIKRPT can be called for any file (RPT1 - RPT7).
- Last call is in effect.
- The number and location of headers are maintained, and reflect the last call.
- You can change the header contents, header order, and number of headers.

Prior to printing the first heading line, QUIKRPT scans each header for the following reserved arguments and replaces the contents as follows:

Argument	Action Taken
$IPLDAT$	Current date in the form of MM/DD/YY
$IPLDYYYY$	Current date in the form of MM/DD/YYYY
$DATE$	Current Julian date in the form of YY.DDD
$JDYYYY$	Current Julian date in the form of YYYY.DDD
$JOBNAM$	Eight-character JOB-NAME
TIM	Current time in the form of HH.MM
PG	Four-digit, zero-suppressed page number
$PAGE$	Seven-digit, zero-suppressed page number

Imperative Functions

LCT Function — Linecount Location

```
CALL QUIKRPT C'LCT' line-loc (C'RPTn')
```

The LCT function defines the location of your LINECOUNT value for the report indicated. The linecount must be in a 2-byte, packed format.

You must establish the linecount value prior to any imperative printing functions. The initial linecount value is saved and used for the subsequent page linecount value.

QUIKRPT decrements the linecount value by the appropriate number for each line printed. When the value is negative, a page eject is performed and page headers are printed.

If omitted, a default value of 60 is used.

SYSnnn Function (VSE Only)

```
CALL QUIKRPT C'SYSnnn' (C'RPTn')
```

The SYSnnn value defines the programmer logical unit SYS-number for the report indicated. The nnn must be specified as three numeric digits. The following SYS-numbers are used as the default:

```
RPT1          SYS005
RPT2          SYS006
RPT3          SYS007
RPT4          SYS008
RPT5          SYS009
RPT6          SYS010
RPT7          SYS011
```

Example of changing RPT1 from the default of SYS005 to SYS021, and RPT2 from SYS006 to SYS022:

```
CALL  QUIKRPT  C'SYS021'                 /* Change RPT1 to SYS021
CALL  QUIKRPT  C'SYS022'  C'RPT2'/* Change RPT2 to SYS022
```

In the QUIKRPT program example shown at the end of this section, the two statements above would be placed immediately before statement 100. SYS021 and SYS022 would then have to be assigned to the appropriate printers in your JCL.

Write Printline Function

```
CALL QUIKRPT printline-loc (C'RPTn')
```

The printline function assumes that, if the first parameter in a call is not any of the valid function codes, the data specified in the first parameter is printed. The first position of the line (printline-loc) must contain a valid ASA control character. The data is assumed to be in positions 2 through 133.

QUIKRPT blanks the entire printline data area (including the first ASA byte) after issuing the print command. The linecount is decremented by 0, 1, 2, or 3 depending upon the ASA control character indicator.

DOHDR Function — Page Headers

```
CALL QUIKRPT C'DOHDR'   (C'RPTn')
```

DOHDR function prints the report headings at the top of the next page. This applies to the RPTn specified, with the appropriate page number incremented and linecount value reset as specified. This operation is not required unless you purposely want to start on a new page.

CLOSE Function

```
CALL QUIKRPT C'CLOSE'  (C'RPTn')
```

QUIKRPT issues a CLOSE macro for this function which is normally requested at end of job. This must be done to ensure proper report file integrity. If a RPTn is specified, a CLOSE for that report file only is performed.

When a CLOSE with no other operand is specified, a CLOSE is issued for all RPTn files used in this particular run.

QUIKRPT Example

The following JCL and VISION:Report statements illustrate the use of QUIKRPT producing two reports, in addition to the detail report printed from VISION:Report. The following numbers correspond to the VISION:Report statement numbers used in the example, and briefly describe the functions in conjunction with QUIKRPT:

Statement Number	Description
20	Read report header data from statements and store in WST area, two statements equal one header line. Report #1 has two headers located at WST1 and WST161. Report #2 has two headers located at WST321 and WST481.

Statement Number	Description
50	Initial linecount values for RPT1 and RPT2. Indicate linecount locations to QUIKRPT, as well as headings.
100	Get input records, process, and print detail report.
150	An exception record is formatted in the PRT area and QUIKRPT is called to print the line to RPT1.
200	Call for closing of reports and EOJ.
300	On a BREAK, a summary total line is printed with QUIKRPT using RPT2, with the line formatted in the SAV area using index pointer PTA.

MVS JCL Example

```
//QUIKRPT JOB  999,REPORT   OPEN ORDER
//STEP    EXEC   PGM=QUIKJOB
//STEPLIB DD DSN=your.rpt.loadlib,DISP=SHR
//SYSPRINT DD    SYSOUT=A     VISION:Report DETAIL REPORT
//RPT1     DD    SYSOUT=A     RPT1 EXCEPTION
//RPT2     DD    SYSOUT=A     RPT2 SUMMARY
//SYSUT1   DD    DSN=ORDMSTR,DISP=SHR
//SYSIN    DD    *
OPTION SEQCHK=NO,PRNTLCT=78
```

VSE JCL Example

```
// JOB QUIKRPT OPEN ORDER REPORTS
    // DLBL filename,'your.rpt.lib'
    // EXTENT SYS001,vvvvvv
    // LIBDEF *,SEARCH=(lib.sublib)
    // ASSGN SYSLST,00E   VISION:Report DETAIL REPORT
    // ASSGN SYS005,01E   RPT1 EXCEPTION
    // ASSGN SYS006,02E   RPT2 SUMMARY
    // ASSGN SYS010,280   INF DEVICE
    // TLBL ORDMSTR,'etc'
    // EXEC QUKBJOB
    OPTION SEQCHK=NO,PRNTLCT=78
    INFTAPE30000100SSYS010   LBL=ORDMSTR
    DETCARD
```

VISION:Report Statements

```
HDR 1A 1 $IPLDAT$         OPEN ORDER DETAIL PAGE$PG$
    HDR 2A  ORDER-NR CUST-NAME          DATE     AMOUNT
    BREAK  1 INF1-6  SB 0 SA 1 PRINT C'**' IN TOT POS 67
    SET PTA WST1                                          /* Set index PTR to WST.
20  GET DET ATEND 50                                      /* Read rpt HDR lines.
    MOVE DET1-80 TO PTA1                                  /* Will be stored in WST.
    SET PTA UP 80                                         /* Bump index by 80.
    GOTO 20                                               /* Go get next HDR.
50  MOVE C'78' TO SAV1-2-P                                /* Lct value for RPT1.
    MOVE C'78' TO SAV3-4-P                                /* Lct value for RPT2.
    CALL QUIKRPT C'LCT' SAV1                              /* Indicate lct.
    CALL QUIKRPT C'LCT' SAV3 C'RPT2'                      /* Lct to RPT1 & RPT2.
    CALL QUIKRPT C'HDR' WST1 WST161                       /* Indicate HDRs.
    CALL QUIKRPT C'HDR' WST321 WST481 C'RPT2'

100 GET INF ATEND 200
    CHECKBREAKS ON BREAK PERFORM 300 THRU 400
    IF INF7-12 IS LT VAL56-61                             /* Test for exception.
         GOTO 150.                                        /* Go process exception.
    ACCUM INF13-18-P IN A 6 BYTE CTA, ON BREAK PRINT IN POS 51 2C
    MOVE INF1-6 TO PRT2                                   /* MOVE data
    MOVE INF21-40 TO PRT12                                /* Fields
    MOVE INF7-12 TO PRT36 F                               /* To
    MOVE INF13-18-P TO PRT51 2C                           /* PRT.
    MOVE INF1-6 TO SAV5-10                                /* Save order
    MOVE INF21-40 TO SAV11-30                             /* And name.
    PRINT                                                 /* PRINT detail line.
    GOTO 100                                              /* Go GET next record.
150 MOVE INF1-6 TO PRT2                                   /* MOVE
    MOVE INF7-12 TO PRT36 F                               /* Record data
    MOVE INF31-40 TO PRT12                                /* Exception
    MOVE INF13-18-P TO PRT51 2C                           /* To PRT area.
    CALL QUIKRPT C'RPT1'                                  /* PRINT to 'RPT1'.
    GOTO 100                                              /* Go GET next record.

200 CALL QUIKRPT C'CLOSE'                                         /* Issue CLOSE.
    GOTO EOJ                                              /* And go to EOJ.

300 IF VAL180 IS EQU TO C'F'                              /* Bypass on final.
         GOTO 400.
    MOVE SPACE TO SAV31-170                               /* Blank SAV area.
    SET PTA SAV32                                         /* Set index PTR.
    MOVE SAV5-10 TO PTA2                                  /* Move summary
    MOVE SAV11-30 TO PTA12                                /* Data to SAV
    MOVE CTA3-8-P TO PTA51 2C                             /* For 'RPT2'.
    MOVE C'0' TO SAV31                              /* ASA control for double-space.
    CALL QUIKRPT SAV31 C'RPT2'                            /* PRINT 'RPT2'.

400 EXIT                                                  /* Exit.
900 END

.  Headers input is entered here.
.
/*
```

QUIKTABL — Automated Tabling Routine

QUIKTABL is a VISION:Report subroutine designed to give extensive table handling capability to MVS and VSE users. The following fully automated functions are supported for up to 16 concurrent tables:

- Table load / reload / delete.
- Sequential retrieval (any starting point).
- Random retrieval.
- Serial search based on any key.
  - Starting from the beginning.
  - Starting at any entry number in the table.
- Binary search based on any key.

Typical applications for QUIKTABL might be:

- Load a description file into a table and retrieve entries by key.
- Build and maintain a bank of accumulators having a one-to-one relationship with some variable such as department number or state code.
- Buffer input records and retrieve later for processing.

QUIKTABL is a 6K routine that is invoked using a CALL statement in procedure logic. You supply a few pieces of information, such as the amount of space for all tables to be handled by QUIKTABL, entry length, and the location of the data to be stored. After the table is loaded, you need only supply the entry number at which to start retrieval, or the key length and location on which to search.

Load a Table

If you want to store entries (fixed length only) in a table, the following procedure should be used:

- Determine the amount of memory that QUIKTABL should require for the table(s). The area allocated must be larger than the number of bytes needed to hold the table entries. This is because of control information used by QUIKTABL. For more information, see Determine the Proper Size of the Table Area.
- Set up the following table control block in a save area or working storage area. The save area is more convenient to use since it does not interfere with other uses of the working storage area.

```
*               SAVE AREA
* TABLE CONTROL BLOCK AREA
EQU  TBL-CNTL-BLK  SAV001-048              /*
EQU  OPCODE        SAV001-004     C'PUT '  /*Table operation
EQU  TABLENUM      SAV005-005-B   X'00'    /*0-15
EQU  ENTRYLEN      SAV006-007-B   X'28'    /*1-32767
EQU  ENTRYNUM      SAV008-009-B   X'01'    /*1-32767
EQU  KEYLENGTH     SAV010-010-B   X'01'    /*1-255
EQU  KEYLOCN       SAV011-011-B   X'01'    /*1-255
EQU  TABLESZE      SAV012-015-B   X'0F48'  /*1024-8M
EQU  RETCODE       SAV016-016-B   X'00'    /*Return code
EQU  MESSAGE       SAV017-048     SPACES   /*Error msg area
```

Where

- OPCODE specifies the function to be done by QUIKTABL. It can have the value of PUT, GET, REPL, FIND, FINR, SRCH, DELT.
- TABLENUM specifies the table number. It can have a value in the range of 0 to 15.
- ENTRYLEN specifies the length of each entry. It can have a value in the range of 1 to 32767.
- ENTRYNUM specifies the entry number to be stored or retrieved. It can have a value in the range of 1 to 32767.
- KEYLENGTH specifies the length of the key in the entry. It can have a value in the range of 1 to 255. A value is required here only for FIND, FINR, or SRCH operations.
- KEYLOCN specifies the location of the key in the table entry. It can have a value in the range of 1 to 255. A value is required only for FIND, FINR, or SRCH operations.
- TABLESZE specifies the size of the table area to be used by QUIKTABL and can have a value in the range of 1024 bytes to 8 megabytes.
- RETCODE holds the return code passed back by QUIKTABL to indicate the success of the request:

Code	Description
0	Successful completion.
1	End of table reached on a retrieval or entry number beyond final table entry.
2	Entry not found (FIND, FINR, SRCH operations).
6	Invalid entry length (PUT operation).
7	Requested table number not found.
8	Table area is full.
9	Invalid / missing parameter or operation code.

- MESSAGE holds an error description returned by QUIKTABL in the event of an unsuccessful operation. To print this message, code procedure logic statements to cause the entries to be loaded into a table. Read the file containing the entries, then invoke QUIKTABL through the CALL facility, passing it three parameters:
  - The field name of the QUIKTABL control block.
  - A literal of 4 zeros (X'00000000').
  - The field name of the entry to be stored.

Example 1

Create a table of 40-byte account number/account name entries. The first five bytes of the entry contain the account number, followed by a 35-byte account name. Store the entries in the table on the second input file. DET is defined in the save area and the table entry is defined in the work area.

```
OPTION SEQCHK=NO
*
* AR FILE
EQU IN-AR-REC       INF001-352
EQU ACCTNO          INF004-008
*
* AR MAILING FILE
EQU IN-M-AR-REC       DET001-352
EQU M-ACCTKEY         DET001-003
EQU M-ACCTNO          DET004-008
EQU M-ACCTNAME        DET050-084
*             WORKING STORAGE AREA
* WST TABLE ENTRY
EQU WORK-AREA-1     WST000-000
EQU WS-ACCTNO       (05)                      /*
EQU WS-ACCTNAME     (35)                      /*
EQU WS-TBL-REC      WST001-040                /* Redefine above flds to tbl ent
*
*               SAVE AREA
* TABLE CONTROL BLOCK AREA
EQU TBL-CNTL-BLK    SAV001-048                /*
EQU OPCODE          SAV001-004      C'PUT '   /*
EQU TABLENUM        SAV005-005-B    X'00'     /* 0-15
EQU ENTRYLEN        SAV006-007-B    X'28'     /* 1-32767
EQU ENTRYNUM        SAV008-009-B    X'01'     /* 1-32767
EQU KEYLENGTH       SAV010-010-B    X'01'     /* 1-255
EQU KEYLOCN         SAV011-011-B    X'01'     /* 1-255
EQU TABLESZE        SAV012-015-B    X'0F48'   /* 1024-8M
EQU RETCODE         SAV016-016-B    X'00'     /*
EQU MESSAGE         SAV017-048      SPACES    /* Error msg area
050  MOVE C'PUT'     TO OPCODE                /* Init tbl cntl blk for load
     MOVE C'0'       TO TABLENUM              /* 1st table
     MOVE C'40'      TO ENTRYLEN              /* Entry length
     MOVE C'1'       TO ENTRYNUM              /* 1st table entry
     MOVE C'3912'    TO TABLESZE              /* Size of table
     MOVE C'0'       TO RETCODE               /* Clr rc
     MOVE SPACES     TO MESSAGE               /* Clr msg
*
100  GET DET ATEND 200                        /* Load table
*
     MOVE M-ACCTNO     TO WS-ACCTNO           /* Move flds to work area
     MOVE M-ACCTNAME   TO WS-ACCTNAME
*
     CALL QUIKTABL TBL-CNTL-BLK X'00000000' WS-TBL-REC
*
     IF RETCODE GT X'00'                      /* If rc not zero
          GOTO 900.                           /*  Tbl load error rtn (&EOJ)
*
     GOTO 100.                                /* Else cont load
*
200  ...CONTINUE PROCESSING...
```

The logic at sequence number 050 shows how the table control block can be initialized. At sequence number 100, the detail file is read until end of file time. Account number and account name are moved to a table entry area in working storage. The call to QUIKTABL loads the next table entry into the table. After each call to QUIKTABL, query the return code to ensure that the desired action took place successfully. Non-zero return codes should be handled as is appropriate for that table operation.

Example 2

This example enhances Example 1 by loading a second table. When handling multiple tables, the same table control block area is used, so special care must be taken that OPCODE, TABLENUM, ENTRYLEN, ENTRYNUM, KEYLENGTH, and KEYLOCN are set correctly for each call to QUIKTABL.

```
OPTION SEQCHK=NO
*
* AR FILE
EQU IN-AR-REC          INF001-352
EQU ACCTNO             INF004-008
*
* AR MAILING FILE
EQU IN-M-AR-REC        DET001-352
EQU M-ACCTKEY          DET001-003
EQU M-ACCTNO           DET004-008
EQU M-ACCTNAME         DET050-084
*
* AR KEY DESCRIPTION FILE
EQU IN-K-AR-REC        INC001-053
EQU K-ACCTKEY          INC001-003
EQU K-DESC             INC004-053
*              WORKING STORAGE AREA
EQU WORK-AREA-1        WST000-000
EQU WS1-ACCTNO         (05)                  /*
EQU WS1-ACCTNAME       (35)                  /*
EQU WS2-ACCTKEY        (03)                  /*
EQU WS2-DESC           (50)                  /*
*
EQU WS1-TBL-REC        WST001-040            /* Redefine above flds to tbl ent
EQU WS2-TBL-REC        WST041-093            /* Redefine above flds to tbl ent
*                 SAVE AREA
* TABLE CONTROL BLOCK AREA
EQU TBL-CNTL-BLK       SAV001-048            /*
EQU OPCODE             SAV001-004     C'PUT '/*
EQU TABLENUM           SAV005-005-B   X'00'  /* 0-15
EQU ENTRYLEN           SAV006-007-B   X'28'  /* 1-32767
EQU ENTRYNUM           SAV008-009-B   X'01'  /* 1-32767
EQU KEYLENGTH          SAV010-010-B   X'01'  /* 1-255
EQU KEYLOCN            SAV011-011-B   X'01'  /* 1-255
EQU TABLESZE           SAV012-015-B   X'1B0A'/* 1024-8M
EQU RETCODE            SAV016-016-B   X'00'  /*
EQU MESSAGE           SAV017-048   SPACES    /* Error msg area
050  MOVE C'PUT'      TO OPCODE              /* Init tbl cntl blk for load
     MOVE C'0'        TO TABLENUM            /* 1st table
     MOVE C'40'       TO ENTRYLEN            /* Entry length
     MOVE C'1'        TO ENTRYNUM            /* 1st table entry
     MOVE C'6922'     TO TABLESZE            /* Size of table
     MOVE C'0'        TO RETCODE             /* Clr rc
     MOVE SPACES      TO MESSAGE             /* Clr msg
*
```

```
100  GET DET ATEND 200                       /* Load table
*
     MOVE M-ACCTNO   TO WS1-ACCTNO           /* Move flds to work area
     MOVE M-ACCTNAME TO WS1-ACCTNAME
*
     CALL QUIKTABL TBL-CNTL-BLK X'00000000' WS1-TBL-REC
*
     IF RETCODE GT ZERO                      /* If rc not zero
         GOTO 900.                           /* Tbl load error rtn (&EOJ)
*
     GOTO 100.                               /* Else cont load
*
200  MOVE C'1'       TO TABLENUM            /* 2nd table
     MOVE C'53'      TO ENTRYLEN            /* Entry length
     MOVE C'1'       TO ENTRYNUM            /* 1st entry
*
250  GET INC ATEND 300                       /* Load table
*
     MOVE K-ACCTKEY  TO WS2-ACCTKEY          /* Move flds to work area
     MOVE K-DESC     TO WST2-DESC
*
     CALL QUIKTABL TBL-CNTL-BLK X'00000000' WS2-TBL-REC
*
     IF RETCODE GT ZERO                      /* IF rc not zero
         GOTO 900.                           /* Tbl load error rtn (&EOJ)
*
     GOTO 250.                               /* Else cont load
*
300  ... CONTINUE PROCESSING ...
```

If you want to retrieve a specific entry by key, starting at the beginning of the table, code the following in your program statements:

- Modify the same control block you used for loading the table:
  - Move a FIND to OPCODE in the control block.
  - If you have not set them up initially in the control block definitions, move the correct values to the KEYLENGTH and KEYLOCN fields.
- Invoke QUIKTABL, passing it four parameters:
  - The field name of the QUIKTABL control block.
  - A literal of 4 zeros (X'00000000').
  - The field name of the area where the found entry is to be placed.
  - The field name of the search key.
- Optionally, check the return code for a found / not found condition:
  - 0 = entry found
  - 2 = entry not found

Example

Using the table created in Example 1 in the previous section, retrieve an account number/account name entry based on a five-byte account number field from the input master file INF. The five-byte account number starts in the fourth byte of the master file. If the entry is found, QUIKTABL moves it to working storage, as specified in the parameter list.

```
200  MOVE C'FIND'    TO OPCODE           /* Init tbl cntl blk for find
     MOVE C'0'       TO TABLENUM         /* 1st table
     MOVE C'40'      TO ENTRYLEN         /* Entry length
     MOVE C'1'       TO ENTRYNUM         /* 1st table entry
     MOVE C'5'       TO KEYLENGTH        /* Length of search key
     MOVE C'1'       TO KEYLOCN          /* Start position of key
     MOVE C'0'       TO RETCODE          /* Clr rc
     MOVE SPACES     TO MESSAGE          /* Clr msg
*
210  GET INF ATEND 990                   /* Read next master rec
*
     CALL QUIKTABL TBL-CNTL-BLK X'00000000' WS-TBL-REC ACCTNO

*
     IF RETCODE EQ X'00'                 /* If entry found
         GOTO 220.                       /*  BR AR
     IF RETCODE EQ X'02'                 /* If entry not found
         GOTO 910.                       /*  Missing entry error msg
     GOTO 920                            /* Else other error msg
*
220  ...CONTINUE PROCESSING
     GOTO 210
```

Retrieve an Entry by Key — Starting at a Specified Entry Number and then Doing a Serial Search

You might want to locate a specific entry by key, but you might not want to start at the beginning of the table. This might be the case if you are retrieving multiple occurrences of entries with the same key, or if you know that a particular entry has to be at or after a certain entry number. Code the following in your procedure statements:

- Modify the same control block you used for creation of the table:
  - Move a FINR (Find Resume) to OPCODE in the control block.
  - If you have not set them up initially in the control block definitions, move the appropriate values to the KEYLENGTH and KEYLOCN fields.
  - Move the appropriate entry number at which to start the search to the ENTRYNUM field.
- Invoke QUIKTABL, passing it four parameters:
  - The field name of the control block.
  - A literal of 4 zeros (X'00000000').
  - The field name of the area where the found entry is to be placed.

  - The field name of the key for which you want QUIKTABL to search.

- Optionally, check the return code in the control block for a found / not found or beyond range condition:
  - 0 = found
  - 1 = beyond end of table
  - 2 = not found

Example

This example uses a table where the entries contain an account number key that is in ascending sequence. The input master file is also in account number sequence. Since the account number is in sequence, use FINR to start a search from the next entry number following the last account number found.

```
200  MOVE C'FINR'    TO OPCODE           /* Init tbl cntl blk for finr
     MOVE C'0'       TO TABLENUM         /* 1st table
     MOVE C'40'      TO ENTRYLEN         /* Entry length
     MOVE C'0'       TO ENTRYNUM         /* 1st table entry, minus 1
     MOVE C'5'       TO KEYLENGTH        /* Length of search key
     MOVE C'1'       TO KEYLOCN          /* Start position of search key
     MOVE C'0'       TO RETCODE          /* Clr rc
     MOVE SPACES     TO MESSAGE          /* Clr msg
*
210  GET INF ATEND 990                   /* Read next master rec
*
     ADD C'1' TO ENTRYNUM                /* Inc table entry number
     CALL QUIKTABL TBL-CNTL-BLK X'00000000' WS-TBL-REC ACCTNO
*
     IF RETCODE EQ X'00'                 /* If entry found
         GOTO 220.                       /* Found match
     IF RETCODE EQ X'02'                 /* If entry not found
         GOTO 910.                       /* Missing entry error msg
     GOTO 920                            /* Else other error msg
*
220  ...CONTINUE PROCESSING
     GOTO 210
```

Retrieve Each Entry Starting from the Beginning

If you want to retrieve each entry in a table, one after the other, code the following in your procedure statements:

- Modify the same control block you used for the loading of the table:
  - Move GET to OPCODE in the control block.
  - If this is the first entry to be retrieved, move 1 to ENTRYNUM in the control block; otherwise, leave the entry number alone. QUIKTABL increments the entry number automatically as it retrieves the entry, so subsequent GET functions retrieve the next entry.

- Invoke QUIKTABL, passing it three parameters:
  - The field name of the control block.
  - A literal of 4 zeros (X'00000000').
  - The field name of the area where the found entry is to be placed.
- Optionally, check the return code in the control block for an end of table condition (RETCODE EQ X'01').

Example

Retrieve each entry from the account number/name table beginning with entry number 1. Have each entry returned to working storage.

Note that moving the 1 to the ENTRYNUM field and the GET to the OPCODE field are one-time-only statements.

```
200  MOVE C'GET'     TO OPCODE              /* Init tbl cntl blk for get
     MOVE C'0'       TO TABLENUM            /* 1st table
     MOVE C'40'      TO ENTRYLEN            /* Entry length
     MOVE C'1'       TO ENTRYNUM            /* 1st table entry
     MOVE C'0'       TO RETCODE             /* clr rc
     MOVE SPACES     TO MESSAGE             /* Clr msg
*
210  CALL QUIKTABL TBL-CNTL-BLK X'00000000' WS-TBL-REC
*
     IF RETCODE EQ X'00'                    /* If entry found
         GOTO 220.                          /*  BR AR
     IF RETCODE EQ X'01'                    /* If end-of-table
         GOTO 300.                          /*  BR to eot proc.
     GOTO 940                               /* Else other error msg
*
220  ...CONTINUE PROCESSING
     GOTO 210
```

Retrieve a Particular Entry by Entry Number

If you want to retrieve a particular entry by entry number in a table, code the following in your procedure statements:

- Modify the same control block you used for the loading of the table:
  - Move GET to OPCODE in the control block.
  - Move the entry number of the entry to be retrieved to the ENTRYNUM field.
- Invoke QUIKTABL, passing it three parameters:
  - The field name of the control block.
  - A literal of 4 zeros (X'00000000').
  - The field name of the area where the found entry is to be placed.

- Optionally, check the return code in the control block for successful completion:
  - 0 = record retrieved
  - 1 = end of table

Example

Retrieve the eighth entry in the account number/name table.

```
MOVE C'GET' TO OPCODE
MOVE C'8' TO ENTRYNUM
CALL QUIKTABL TBL-CNTL-BLK X'00000000' WS-TBL-REC
```

Binary Search for a Particular Entry by Key

If you have a table of 20 entries or more that are in ascending sequence by key, you could perform a binary instead of a serial search. For such a table, a binary search is much faster than a serial search. Code the following in your procedure statements:

- Modify the same control block you used for the loading of the table:
  - Move SRCH to OPCODE in the control block.
  - If you have not set them up initially in the control block definitions, move the correct values to the KEYLENGTH and KEYLOCN fields.
- Invoke QUIKTABL, passing it four parameters:
  - The field name of the QUIKTABL control block.
  - A literal of 4 zeros (X'00000000').
  - The field name of the area where the found entry is to be placed.
  - The field name of the key in the record (not entry) to be retrieved.
- Optionally, check the return code in the control block for a found/not found condition:
  - 0 = found
  - 2 = not found

Example

Retrieve a particular entry by account number key using the binary search technique. The search key is account number.

```
200  MOVE C'SRCH'    TO OPCODE              /* Init tbl cntl blk for bin srch
     MOVE C'0'       TO TABLENUM            /* 1st table
     MOVE C'40'      TO ENTRYLEN            /* Entry length
     MOVE C'1'       TO ENTRYNUM            /* 1st table entry
     MOVE C'5'       TO KEYLENGTH           /* Length of search key
     MOVE C'1'       TO KEYLOCN             /* Start position of key
     MOVE C'0'       TO RETCODE             /* Clr rc
     MOVE SPACES     TO MESSAGE             /* Clr msg
*
210  GET INF ATEND 990                      /* Read next master rec
*
     CALL QUIKTABL TBL-CNTL-BLK X'00000000' WS-TBL-REC ACCTNO
*
     IF RETCODE EQ X'00'                    /* If entry found
         GOTO 220.                          /*  BR AR
     IF RETCODE EQ X'02'                    /* If entry not found
         GOTO 910.                          /*  Missing entry error msg
     GOTO 920                               /* Else other error msg
*
220  ...CONTINUE PROCESSING...
     GOTO 210
```

Replace an Entry in a Table

If you want to update a particular entry in a table, code the following in your procedure statements:

- Modify the same control block you used for loading the table:
  - Move REPL to the OPCODE field.
  - Move the entry number of the entry you want replaced to the ENTRYNUM field, if it does not already contain it.
- Invoke QUIKTABL, passing it three parameters:
  - The field name of the QUIKTABL control block.
  - A literal of 4 zeros (X'00000000').
  - The field name of the entry that is to replace the old entry.
- Optionally, check the return code field in the control block for successful completion:
  - 0 = successful completion.
  - 1 = entry number greater than number of entries in table.

Example

Each entry in a table contains a three-byte packed counter starting in location 41-43 (WSCOUNTER). A particular entry's counter field is incremented by 1 if its key (location 1) matches a key contained in the first five bytes of the input record (ACCTNO). In order to update the entry, it is first retrieved by the FIND operation code; then it is replaced after the counter has been incremented:

```
*             WORKING STORAGE AREA
* WST TABLE ENTRY
EQU WORK-AREA-1     WST000-000
EQU WS-ACCTNO       (05)                    /*
EQU WS-ACCTNAME     (35)                    /*
EQU WS-COUNTER      (03)-P                  /*
EQU WS-TBL-REC      WST001-043              /* Redefine above flds to tbl ent
*
190  MOVE C'0'       TO TABLENUM            /* 1st table
     MOVE C'43'      TO ENTRYLEN            /* Entry length
     MOVE C'1'       TO ENTRYNUM            /* 1st table entry
     MOVE C'5'       TO KEYLENGTH           /* Length of search key
     MOVE C'1'       TO KEYLOCN             /* Start position of search key
*
200  MOVE C'FIND'    TO OPCODE              /* Init tbl cntl blk for find
     MOVE C'0'       TO RETCODE             /* Clr rc
     MOVE SPACES     TO MESSAGE             /* Clr msg
*
210  GET INF ATEND 990                      /* Read next master rec
*
     CALL QUIKTABL TBL-CNTL-BLK X'00000000' WS-TBL-REC ACCTNO
*
     IF RETCODE EQ X'00'                    /* If entry found
         GOTO 220.                          /* BR AR
     IF RETCODE EQ X'01'                    /* If entry not found (eot)
         GOTO 910.                          /* Missing entry error msg
     GOTO 920                               /* Else other error msg
*
220  ADD C'1' TO WS-COUNTER                 /* Inc ctr for this acct nbr
*
     MOVE C'REPL'    TO OPCODE              /* Init tbl cntl blk for replace
*
     CALL QUIKTABL TBL-CNTL-BLK X'00000000' WS-TBL-REC
*
     IF RETCODE GT X'00'                    /* If entry not replaced
         GOTO 940.                          /* Then replace error
*
230  ...CONTINUE PROCESSING
     GOTO 200
```

Delete a Table

You might want to delete a table in order to rebuild the table or to free up space in the table work area. To delete a table, code the following in your procedure statements:

- Modify the same control block you used for the loading of the table:
  - Move a DELT to the OPCODE field of the QUIKTABL control block.

- Invoke QUIKTABL, passing it three parameters:
  - The field name of the QUIKTABL control block.
  - A literal of 4 zeros (X'00000000').
  - Any valid subparameter to satisfy QUIKTABL's parameter requirement.

Example

Delete table number 1 in order to free up the space in the table area. Assume the table number is already in the table number field of the control block.

```
MOVE 'DELT' TO OPCODE
CALL QUIKTABL OPCODE X'00000000' OPCODE
```

User Error Checking and Handling

QUIKTABL indicates the result of each request in the return code (RETCODE) field of the control block area. Check this return code following each CALL to QUIKTABL.

06 - Invalid Entry Length.

Message	Description
Cause	On a PUT (load) operation, the ENTRYLEN field in the control block contains a value less than 1 or greater than 32767.
Result	The PUT is not performed and QUIKTABL returns a message in the MESSAGE field of the control block, along with a return code of 6 in the RETCODE field. If you ignore the error and repeat the operation, the program continues returning a 6.
Action	You should correct the entry length value and rerun the program.

07 - Requested Table Not Found.

Message	Description
Cause	A retrieve, replace, or delete operation is being specified for a table that does not exist.

Message	Description
Result	The requested operation is not performed and QUIKTABL returns a message in the MESSAGE field of the control block, along with a return code of 7 in the RETCODE field. If you ignore the error and repeat the operation, the program continues returning a 7.
Action	You should provide the correct table number and rerun the program.

08 - Tabling Area is Full.

Message	Description
Cause	A PUT operation has been requested, but there is insufficient room in the table area.
Result	The entry is not stored and QUIKTABL returns a message in the MESSAGE field of the control block, along with a return code of 8 in the RETCODE field. If you ignore the error and attempt to store another entry, QUIKTABL abends with a data exception.
Action	You should allocate additional table space by increasing the amount in the TABLESZE field of the QUIKTABL control block. You should then rerun the program.

09 - Invalid/Missing Parameter or Opcode.

Message	Description
Cause	One or more of the required parameters for a given operation code, or the operation code itself, is either missing or invalid.
Result	QUIKTABL abends with a data exception.
Action	You should check the CALL statement(s) invoking QUIKTABL and ensure that the correct number of parameters were passed and that the parameters specified the correct areas for the operation requested. Also, you should check the QUIKTABL control block to make sure that the correct values or defaults were present when the call to QUIKTABL was executed.

Determine the Proper Size of the Table Area

The size of the table work area is determined chiefly by the amount of data you want to store. However, additional data required for table management is also placed in the table area by QUIKTABL, so the allocation must be greater than the number of bytes required for user entries. Allow for a 512-byte control block and two bytes per 510 bytes of tabling. At least 1024 bytes of work area must be allocated. The following table is provided as a guide in deciding how large to make the table area.

Table Size	Max. Data Capacity
1024	510
2048	1530
4096	3570
8192	7650
16384	15810
32256	31620
65536	64260
31072	129030
262144	258570

QUIKTIME — Time Subroutine

QUIKTIME returns the current time in HHMMSS and in units of 1/300 seconds. Both fields are packed decimal.

```
CALL QUIKTIME RETURN-AREA
```

Term	Description
RETURN-AREA	Either a valid VISION:Report field definition or an equated data name. The field must be 12 bytes long. After control is returned from QUIKTIME, the format of return-area is: ■ Bytes 1-4 packed decimal = HHMMSS time format ■ Bytes 5-12 packed decimal = units of 1/300 seconds

Example

The following example calls QUIKTIME ten times and prints the time formats returned.

```
OPTION LIST=YES,SEQCHK=NO
EQU WST-AREA       WST
EQU TIME-HHMMSS    (4)-P
EQU TIME-1_300-SEC (8)-P
EQU LOOP-COUNTER   (2)-P    ZEROES
TITLE 'TIME FORMATS RETURNED BY QUIKTIME'
REPORT TIME-HHMMSS TIME-1_300-SEC
010 IF LOOP-COUNTER EQ P'10'
       GO TO EOJ.
    CALL QUIKTIME TIME-HHMMSS
    PRINT REPORT
    ADD C'1' TO LOOP-COUNTER
    GO TO 010
999 END
```

QUIKTRAN — ASCII/EBCDIC Translator

Many installations have a recurring or occasional requirement to translate the data representation of a file (that is, ASCII to EBCDIC or EBCDIC to ASCII). The QUIKTRAN routine, invoked by a single CALL statement, provides you with the ability to automatically translate to other representations.

The translate table module/phase defaults are QUBKTRN for VSE and QUIKTRNT for MVS. See QUIKTRNT —Translate Table (MVS Only) QUKBTRN —Translate Table (VSE Only) for details.

File Data Translation Routine

```
005 MOVE C'200' TO WST1-2-B              /* Make record length avail to QUIKTRAN.
010 GET                                  /* GET the next recd to be translated.
    CALL QUIKTRAN C'ASCII-TO-EBCDIC' INF1 WST1-2  /* Translate record.
    MOVE INF1-200 TO OFA1-200            /* MOVE translated recd to output area.
    WRITE OFA                            /* WRITE the output record.
    GOTO 010                             /* GO GET and translate next record.
```

```
                OP1                    OP2      OP3
CALL QUIKTRAN   C'ASCII-TO-EBCDIC'     FLDDEF    FLDDEF
                C'EBCDIC-TO-ASCII'
```

Each execution of a CALL QUIKTRAN causes the translation of one block of data. The following coding illustrates the context in which QUIKTRAN would be used.

Operands

- If the input data is ASCII and is to be translated to EBCDIC, code C'ASCII-TO-EBCDIC'. If the input is EBCDIC and is to be translated to ASCII, code C'EBCDIC-TO-ASCII'. If a data representation other than ASCII or EBCDIC is involved, this routine must be customized at installation. Refer to the installation materials for customization instructions or to the systems programmer who installed VISION:Report for other translation representations.

  Note: Some EBCDIC values have no direct translation into ASCII. In this case, QUIKTRAN translates the character to a null (hex 00) character. For this reason user caution is advised.

- Code a VISION:Report field definition for the leftmost byte of the block of data to be translated. As illustrated in the example above, the current INF record would be the most common block of data, but any VISION:Report area can be specified.

- Define a two-byte binary field containing the length of the data block to be translated each time the call is executed. This length can be any number up to 65535. For fixed length files, this requirement can be satisfied by setting up a constant as in statement 005 of the example above. For variable length files or unusual cases, this value can be calculated before each call.

Some areas should not be translated (such as, binary fields and the LLbb of variable length records).

```
005 MOVE C'200' TO WST1-2-B              /* Make record length avail to QUIKTRAN.
010 GET                                  /* GET the next recd to be translated.
    CALL QUIKTRAN C'ASCII-TO-EBCDIC' INF1 WST1-2  /* Translate record.
    MOVE INF1-200 TO OFA1-200            /* MOVE translated recd to output area.
    WRITE OFA                            /* WRITE the output record.
    GOTO 010                             /* GO GET and translate next record.
```

QUIKTRNT —Translate Table (MVS Only)
QUKBTRN —Translate Table (VSE Only)

The two tables, QUIKTRNT and QUKBTRN, are similar.

QUIKTRNT and QUKBTRN contain two 256-byte tables. The first table translates data displayed in the first (character) line of a PRINTHEX or PRINTCHAR output. The second table translates data in the second (zone) and the third (numeric) lines of a PRINTHEX output.

Execution of PRINTHEX or PRINTCHAR uses the default table. If any alternate tables have been installed in your system, you must identify the alternate table to VISION:Report by OPTION TRLNAME= user-table-name.

QUIKVEQU — EQU Statements for VAL Area

In VISION:Report, a table of EQU statements for the VAL area is automatically loaded during the compile phase. These EQU statements could have been modified. Check with your system programmer for a printout of the new EQU statements.

See the section EQU in Chapter 3 for a list of the default EQU statements supplied for the VAL area.

Note: The table QUIKVEQU is for MVS and VSE. Prior to Release 16.0, it was QUIKVEQU for VSE.

QUKBLIB — VSE Library Interface (VSE Only)

The VSE Library program, QUKBLIB, is in the object and phase library which was loaded when VISION:Report was installed. This is a VISION:Report callable routine to allow sequential and random retrieval of VSE library members in a read-only mode. QUKBLIB provides the VSE user with a convenient method of retrieving one or more members from the library. Sequential and random retrieval of library members are supported. Generic retrieval of members (generic mode) is also permitted, and is specified by the usage of an asterisk (*) within the member name or member type.

QUKBLIB is only supported in VSE environments that support the VSE LIBRM macros facilities.

The following JCL for the QJLIB library containing 'A.' and 'Q.' members is assumed in describing the various calling parameters. Members QUIKDATE.A, and other members starting with the prefix of QUIK, as well as AMORTIZE.Q are in the library.

```
// DLBL  QJLIB,'QJ.LIB'
// EXTENT   ,DOS004
// LIBDEF   SOURCE,SEARCH=QJLIB.QJ150
```

The CALL statement presents basic parameter information to QUKBLIB. The passing parameters can be either a literal or an equated field-name, although it can be shown in either format. It is the function of the library subroutine to interpret the parameters and build and issue the necessary I/O operations to satisfy the requested functions. Typically, the user has to specify the following:

Operands

Term	Description
Operation Code	8-byte field. What function or command is to be performed (get a record, open a file, close a file.)
Library-Name	8-byte file name on the DLBL statement, padded with spaces to the right as necessary (such as, QJLIB).
Sub-Library-Name	8-byte sub-library name, padded with spaces to the right, as necessary (such as, SOURCE).
Member-Name	8-byte member name that you want to start processing, padded with spaces to the right as necessary. If an asterisk is found, the generic mode is set. For example: ■ QUIKDATE to start processing this member. ■ 'QUIK*' to process all members with the prefix QUIK. ■ '*' to process all members.

Term	Description
Member-Type	8-byte member type, padded with spaces to the right, as necessary. If an asterisk is found, the generic mode is set. For example: ■ 'Q' to start processing all member types of Q. ■ '*' to process all member types.
Chain ID	8-byte chain ID, padded with spaces to the right, as necessary (such as, SOURCE).
Feedback-Area	132-byte feedback area. This is the communications area, providing information on any possible errors, and is established by the OPEN command. It consists of: ■ 2-byte binary — Reason code ■ 2-byte-binary — Return code ■ 8-byte — Operation code causing error ■ 120-byte — Error message Refer to the IBM Systems Macros Reference Guide, "Librarian Feedback Codes" for further details. If VAL46-49 (VISION:Report's own return code area) is non-zero, check the above error message and codes in order to diagnose the error further. The 8-byte operation code can detail the exact I/O operation that was being performed when the error occurred.
Record-Area	Work area for record to be read into.
New Member-Name	8-byte new member-name. This is available only when both member-name and member-type are '*' (generic mode). This field is filled with the next member name when the last record of the current member has been read; in the event of an EOF condition, this field is filled with high values. This field is required even when not in generic mode.
New Member-Type	8-byte new member-type. This is available only when both member-name and member-type are '*' (generic mode). This field is filled with the next member type when the last record of the current member has been read; in the event of an EOF condition, this field is filled with high values. This field is required even when not in generic mode.

Not all calls require each operand or parameter to be passed. The specific order of the passing parameters must be adhered to. Not all passing parameters are validated for correctness.

The return codes in the feedback area are general in nature, with the higher return codes usually more serious. If these return codes are 8 or higher, consideration should be given to terminate the program. QUKBLIB adds its own return codes in an effort to assist in pinpointing the problem.

The return codes in the feedback area are:

Code	Description
0000	Successful completion. The service worked as requested.
0004	The requested function was performed, but an exceptional condition exists, or the function was not performed, because the requested result already exists.
0008	Some functions are not or only partially executed.
0012	The requested service could not be performed at all, because the addressed library resource was not available.
0016	There is an externally controllable condition, such as lack of resources or storage space, which resulted in the failure. The return code is accompanied by a librarian message.
0020	An error condition has occurred as a result of internal librarian processing. The return code is accompanied by a librarian message.
0032	Unauthorized access to a library object. The return code is accompanied by a librarian message of L163I.

If a message is generated by the librarian as a result of return code 16 or higher, it is passed back in the error message area of the feedback area.

Opening a Library

```
CALL   QUKBLIB   C'OPEN'   C'library-name'  C'sub-library name'
       C'member-name'  C'member-type'  C'chain-id'      FEEDBACK-AREA
```

- Only one library can be opened and processed at a time.
- This CALL is required and must be the first CALL to QUKBLIB prior to any processing. The feedback area is only established with the OPEN command, as is the generic mode for member-name and member-type. Therefore, as an example, if you want to change the feedback area or switch from generic mode, you must issue a CLOSE command, and reestablish the feedback area and/or generic mode with an OPEN.
- Internally, QUKBLIB issues several I/O operations to verify that the library, sublibrary, and member name and type exists. In the event of an error, the operation code in the feedback area contains the last attempted I/O operation. As an example, STATMBR would signify that a verification on the member was attempted and failed.

- Examples:

```
CALL  QUKBLIB  C'OPEN'    C'QJLIB      '        C'QJ150     '
       C'AMORTIZE'  C'Q        '      C'SOURCE    ' FB-AREA

CALL  QUKBLIB  C'OPEN ' LIB-NAME    SUBLIB-NAME
      C'QUIK*       '   C'A         '    CHAIN-NAME     FEEDBK-AREA
```

VAL 46-49 Return Codes:

Code	Description
0000	Call was successfully completed. The service worked as requested.
0100	Missing or bad parameters passed. Check parameters to ensure that the sequences are correct, or if proper spaces are required for certain passing parameters.
0108	Library could not be found. Check parameters for valid library names, sublibrary, or chain ID. Asterisks (*) are not permitted for these parameters.
0112	Member could not be found. Check parameters for valid member name and type, library names, sublibrary, or chain ID.
0116	Virtual storage could not be obtained. Check SIZE parameters on your EXEC statement and increase the availability of your GETVIS area.
0120	Library could not be opened. Most likely, the member name and/or member type could not be found. Check librarian feedback codes, as well as parameters for valid member name and type, library names, sublibrary, or chain ID.

If VAL46-49 is non-zero, check the feedback area return codes.

Retrieve a record

```
CALL QUKBLIB  C'GET'              RECORD-AREA
     MEMBER-NAME                  MEMBER-TYPE
     NEW-MEMNAME                  NEW-MEMTYPE
```

The record is retrieved and placed into the 80-byte record area. Upon reaching the end of the current member, the next member name and type is placed into the new member name and new member type fields, respectively, if an asterisk has been specified in member-name and member-type (generic mode). If the end of file (EOF) is reached, high values (X'FF') are placed in the new member name and new member type fields.

A new member name and/or member type can be requested by the user, even before the current member reaches end of member.

End of Member Processing

Upon reaching the end of the current member, the next member name and type is placed into the new member name and new member type fields respectively, if an asterisk has been specified in member-name and member-type (generic mode). A return code of 0004 is placed in the VAL46-49 area, and the operation code in the feedback area will contain 'EOM/TYP'.

The last record is moved into the user's record area. If the end of file (EOF) is reached, high values (X'FF') are placed in the new member name and new member type fields, along with the last record being moved into the user's record area.

Upon reaching the end of the current member, or a GET command is requested for another member (even before reaching end of member), QUKBLIB issues the internal I/O commands necessary in an attempt to accommodate the request. These I/O operations include the CLOSE and OPEN commands. If any errors occur, examine the VAL46-49 return codes and the error message and return codes for those commands.

VAL46-49 Return Codes:

Code	Description
0000	Call was successfully completed. The service worked as requested.
0004	End of member. The record in the record-area is the last record for this member and/or type. The operation code in the feedback area should have 'EOM/TYP".
0008	End of File. The record in the record-area is the last record for this library. Any further attempts to read the record will probably cause an abend. The operation code in the feedback area should have EOF.
0100	Missing or bad parameters passed. Check parameters to ensure that the sequences are correct or if proper spaces are required for certain passing parameters.
0112	Member could not be found. Check parameters for valid member name and type, library names, sublibrary, or chain ID.
0150	Length of record is not 80 bytes. Library format or type is invalid.
0154	Undeterminable error. Check feedback area and codes, as well as parameters for valid member name and type.

If VAL46-49 is non-zero, check the feedback area return codes.

Closing a Library

```
CALL  QUKBLIB  C'CLOSE'
```

Along with closing the library previously opened, this CALL frees buffer areas and resets the feedback area pointers and generic mode settings. This CALL should always be made before going to end of job (EOJ). If the library has not been opened, this command is ignored.

VAL46-49 Return Codes:

Code	Description
0000	Close was successfully completed.
0004	There was at least one additional parameter coded; this is not needed and should be corrected, but the CLOSE was successfully completed.

If VAL46-49 contains other than the above codes, check the feedback area return codes.

Miscellaneous Return Codes

VAL46-49 Return Codes:

Code	Description
0200	The function or command requested is invalid or non-existent. Check spelling.
0204	This error should not occur. The pointer to the feedback area is invalid or contains zeros. Normally, after an OPEN, the feedback area has been established. Contact Technical Support (see the section Contacting Computer Associates in Chapter 1.)
0208	Library has not been opened. Function requested cannot be performed.

TOTAL Interface (VSE Only)

VISION:Report access to TOTAL database files is accomplished by the CALL verb, in a manner similar to COBOL. The following VISION:Report statements are examples of such a call to TOTAL:

```
010 CALL QJDATBAS C'SINON' WST101 C'UPDATE' C'BASE01' C'END.'
150 CALL QJDATBAS C'READV' WST1 USER-PARMS etc etc C'END.'
```

QJDATBAS is a callable routine which can be renamed. Data areas and/or literals can be used in combination in the parameter list to communicate requests. The parameters specified in the QJDATBAS call list are the same as other languages would use for calls to DATBAS (see the TOTAL Guide).

Two requirements must be met to use TOTAL database files successfully with VISION:Report as shown above.

- A callable subroutine in the phase library, which includes DATBAS (or DATBAS7).
- Reserved core immediately following DATBAS for the TOTAL package to use for program and I/O buffers.

See the installation materials for additional information.

TOTAL4 Interface

VISION:Report access to TOTAL database files is accomplished by the CALL verb, in a manner similar to COBOL. The following VISION:Report statements are examples of such a call to TOTAL:

```
CALL QJTOTAL C'OPENM' WST101 C'MMMM' C'END.'
CALL QJTOTAL C'SEQRM' WST1 C'MMMM' C'MMMMCTRLMMMMDATA' WST100 C'END.'
```

QJTOTAL is a callable routine which can be renamed by you. Data areas and/or literals can be used in combination in the parameter list to communicate requests. The parameters specified in the QJTOTAL call list are the same as other languages would use for calls to TOTAL (see the TOTAL Guide).

The following requirement must be met to successfully use TOTAL database files with VISION:Report as shown above.

A callable subroutine in the phase library, which includes TOTAL4, the appropriate database module, and direct access module.

Refer to the *VISION:Report Installation Guide* for additional information.

Chapter

7

QUIKVSAM

QUIKVSAM

QUIKVSAM provides you with native mode access to VSAM data sets (files), enabling you to support commonly used VSAM functions with the minimum of effort.

The QUIKVSAM subroutine is automatically installed as part of the VISION:Report installation. In one execution of VISION:Report, QUIKVSAM can access a total of 256 concurrently open files, plus the 53 files available through standard VISION:Report. QUIKVSAM can also be called from Assembler or COBOL language programs.

QUIKVSAM provides you with the capabilities to process keyed sequenced (KSDS), entry sequenced (ESDS), and relative record (RRDS) data sets in either a random or sequential mode. In addition, LDS, VRRDS (OS/390 only) and VRDS (VSE only) files are supported. Records can be of a fixed, variable, or spanned format. KSDS files can have an LRECL greater than 32K. Functional commands (operands) are available to retrieve, update, insert, load, and delete records. An extensive diagnostic/error display routine is included to assist you in debugging illogical requests or determine the cause of abnormal conditions.

The user ABEND code 3998 is issued in the event an error occurs from which QUIKVSAM cannot recover.

Note: Prior to Release 16.0, you could not mix native VSAM coding with calls to QUIKVSAM.

Prerequisites

- System generated with VSAM support.
- VSAM and Access Method Services (AMS) modules in appropriate library.
- A master and/or user catalog.
- Any VSAM data set (cluster, path, alternate index) referenced by QUIKVSAM must have previously been defined using AMS (IDCAMS).

- Partition or region size large enough to accommodate VISION:Report and VSAM routines (512K or larger).

Application

The CALL statement presents basic parameter information to QUIKVSAM. It is the function of the VSAM subroutine to interpret the parameters and build and issue the necessary I/O operations to satisfy the requested functions. Typically, you have to specify the following:

- The ddname (file name) of the data set to access. The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions). Be sure to include any spaces necessary for padding to bring it to the correct length.
- Do not use CLOSE for DDname or filename.
- What function is to be performed (such as, get a record, put a record, point, update, erase).
- The location of the logical record for the specified function.
- The location of the key of the logical record for the specified function.
- The optional parameters providing for type of record matches, communications/feedback, passwords, and closing of data sets.

Function	Operand One	Operand Two	Operand Three	Operand Four	Operand Five	Operand Six	KSDS	ESDS	RRDS	VRDS	LDS
Retrieve/ Sequential	CALL QUIKVSAM	C'ddname'	C'GET' or C'SEQNTL'	Rec-Area			X	X			X
Retrieve/ Random	CALL QUIKVSAM	C'ddname'	C'READ' or C'RANDOM'	Rec-Area	Key-Area	[C'KEQ' C'KGE']	X	X	X	X	X
Retrieve for Update Sequential	CALL QUIKVSAM	C'ddname'	C'GET-UPD'	Rec-Area			X	X			X
Retrieve for Update Random	CALL QUIKVSAM	C'ddname'	C'READ-UPD'	Rec-Area	Key-Area	[C'KEQ' C'KGE']	X	X	X	X	X
Update/Change (Sequential or Random)	CALL QUIKVSAM	C'ddname'	C'UPDATE'	Rec-Area			X	X	X	X	X
Load/Insert Sequential	CALL QUIKVSAM	C'ddname'	C'LOAD'	Rec-Area			X	X			X
Add/Insert Random	CALL QUIKVSAM	C'ddname'	C'ADD'	Rec-Area			X				

Function	Operand One	Operand Two	Operand Three	Operand Four	Operand Five	Operand Six	KSDS	ESDS	RRDS	VRDS	LDS
Point Generic Position	CALL QUIKVSAM	C'ddname'	C'POINT' C'PNT'	Key-Area	[C'KGEnn' C'KEQ']		X	X	X	X	X
Erase (Seq. & Ran.)	CALL QUIKVSAM	C'ddname'	C'ERASE'				X		X	X	
Communication/ Feedback Area	CALL QUIKVSAM	C'ddname'	C'OPTION' C'OPT-RESET'	Opt-Area Opt-Area	[C'password'] [C'password']		X	X	X	X	X
Open	CALL QUIKVSAM	C'ddname'	C'OPEN'	Opt-Area	[C'password']		X	X	X	X	X
Close	CALL QUIKVSAM	C'ddname'	C'CLOSE'				X	X	X	X	X
Close All Data Sets	CALL QUIKVSAM	C'CLOSE'					X	X	X	X	X
Temporary Close	CALL QUIKVSAM	C'ddname'	C'TCLOSE'				X	X	X	X	X
Temporary Close New File	CALL QUIKVSAM	C'ddname'	C'CLOSER'				X				
Retrieve/ Sequential	CALL QUIKVSAM	C'ddname'	C'RRGET'	Rec-Area	Key-Area				X	X	
Retrieve for Update Sequential	CALL QUIKVSAM	C'ddname'	C'RRGET-UPD'	Rec-Area	Key-Area				X	X	
Load/Insert Sequential	CALL QUIKVSAM	C'ddname'	C'RRLOAD'	Rec-Area	Key-Area				X	X	
Add/Insert Random	CALL QUIKVSAM	C'ddname'	C'RRADD'	Rec-Area	Key-Area				X	X	

Legend:

[ ] = Optional Parameter
ddname = ddname or file name
Opt-Area = Optional Feedback Area
Key-Area = Key Search Argument Area
Rec-Area = Record Return Area

QUIKVSAM Communication/Feedback Area Contents

Position	Definition	Possible Value to Expect	Format
1	File Type	K = KSDS E = ESDS R = RRDS	EBCDIC
2	Type Access	C = Base Cluster P = Path X = Alternate Index	EBCDIC
3	Reserved	Not Used	
4-7	Record Length	Length of last record retrieved or length of record to be added or updated	Binary
8	VSAM Return Code (RC)	Severity of error X'00' = Successful operation X'04' = Warning, possible error X'08' = Failure, logical error; also end-of-file or no-record-found X'0C' = Failure, physical I/O error X'0F' = QUIKVSAM detected illegal call; see error message on diagnostic printout	Binary
9	VSAM Error Code (EC)	X'00' = Successful operation X'04' = End-of-file detected X'08' = Duplicate record X'0C' = Record out of sequence X'10' = No-record-found Any Other = Causes QUIKVSAM to ABEND with diagnostic routine	Binary
10-13	Relative Byte Address (RBA)	RBA of the last record retrieved, added, or updated	Binary
Or 10-17	Extended Relative Byte Address (XRBA)	XRBA of the last record retrieved, added, or updated	Binary

QUIKVSAM Description

The VSAM subroutine examines the parameters presented to it on each CALL request. The degree of examination is limited to checking for too many/too few operands for the action requested.

It is your responsibility to assure the validity of information such as, data integrity, adequate space for records, valid record lengths, inclusion of keys where required. Any violation detected by VSAM results in an aborted operation and the appropriate RC/EC (return/error codes).

The following notes are applicable to QUIKVSAM usage:

- When any UPDATE, ADD, or LOAD function occurs on a data set, an OPTION or OPT-RESET request is required. This request must be the first CALL request made to the data set. *The use of the OPTION request automatically opens the VSAM file in UPDATE mode.* OPT-RESET will additionally reset the most frequently used RBA or XRBA to zero if the data set has been defined with the REUSE parameter. This is required for communicating the record length in the feedback area and allows for possible output processing. The number and size of VSAM buffers used are the same as those specified in the JCL or catalog, or the default taken by VSAM.
- When doing retrieval, an area must be available for the length of the maximum record size. Maximum record size is the one defined in the VSAM catalog. Any records not of maximum record size will have the remaining portion of the area blanked.
- When doing retrieval, the record return area contains high values (X'FF') when end of data is reached or on a no-record-found condition. If the options feedback area is used, these condition codes are also present in the RC/EC fields.

KSDS

Record sizes in a KSDS can be changed in the update function. The key of the record cannot be changed. Records must adhere to specifications as they are defined in the catalog.

A record can be deleted (erased). The only requirement is that it must have been retrieved for update (random or sequential) immediately prior to the ERASE function.

ESDS

Any direct/random access requires a key as noted in the CALL formats. The key is the RBA (relative byte address) of the record. It is always specified in a four-byte binary format. The RBA of retrieved records is available in positions 10-13 of the feedback area.

Records added to an ESDS file must be done in sequential (LOAD) mode. Each record is written in the next available entry position. ESDS record sizes cannot be changed once they have been added to the file.

Deleting (erasing) records in an ESDS file is not allowed.

RRDS

Relative record usage has a unique key that requires special handling. The key is the relative record occurrence (or slot number); however, it is not required to be contained in the data record itself. So the QUIKVSAM subroutine has been designed to use RRDS with a required operand for key location.

When doing random mode processing with RRDS files, you must supply the key for the record wanted. When doing sequential mode processing with RRDS files, the key of the record is supplied to the user in the key area after the operation is completed. The key or relative record number in all cases is in a 4-byte binary format.

QUIKVSAM Level of Support

QUIKVSAM Level of Support	KSDS (Keyed)	ESDS (Entry)	RRDS (Relative Record)
Processing Mode	Random / Sequential	Random / Sequential	Random / Sequential
Type of access	Key	Address (RBA) or XRBA	Key (Relative Record #)
Alternate indexes	Yes	Yes	Not allowed
KEQ / KGE search option (key equal, key greater or equal	Yes	Yes	Yes
Backward processing	No	No	No
Reset at open to null file (to use as a work file)	Yes	Yes	Yes
Key length	User defined	4-Bytes Binary (RBA)	4-Bytes Binary (Relative Record #)
VSAM return code	Position 8 of feedback area	Position 8 of feedback area	Position 8 of feedback area
VSAM error	Position 9 of feedback area	Position 9 of feedback area	Position 9 of feedback area
RBA (Relative Byte Address)	Positions 10-13 of feedback area	Positions 10-13 of feedback area	Positions 10-13 of feedback area
XRBA (Extended Relative Byte Address) **Note:** Supported only by OS / 390	Positions 10-17 of feedback area	Positions 10-13 of feedback area	Positions 10-13 of feedback area
Record length	Positions 4-7 of feedback area	Positions 4-7 of feedback area	Positions 4-7 of feedback area

VSAM Function/Option

VSAM Function/Option		QUIKVSAM Command Name (KSDS)	QUIKVSAM Command Name (ESDS)	QUIKVSAM Command Name (RRDS)
Simple Record Retrieval	Sequential	GET	GET	RRGET
	Random	READ	READ	READ
Update Record Retrieval	Sequential	GET-UPD	GET-UPD	RRGET-UPD
	Random	READ-UPD	READ-UPD	READ-UPD
Add Records	Create/Insert Sequential	LOAD	LOAD	RRLOAD
	Add/Insert Random	ADD	Not allowed	RRADD
Update Records	Random and Sequential	UPDATE	UPDATE Cannot change record length. This is a VSAM restriction on ESDS and RRDS data sets.	UPDATE
Delete Records	Random and Sequential	ERASE	Not allowed	ERASE
Positioning	Generic Point	POINT	POINT	POINT
Housekeeping	Feedback Area	OPTION/ OPT-RESET	OPTION/ OPT-RESET	OPTION/ OPT-RESET
	Data Set Close	CLOSE	CLOSE	CLOSE
	Data Set Close and Reopen	CLOSER	Not applicable	Not applicable
	Data Set Temporary Close	TCLOSE	TCLOSE	TCLOSE

VSAM Share Options

This section contains information on VSAM share options which is essential if you need to reference a data set with more than one ddname or have multiple partition/region considerations.

Definition: The level of sharing allowed for accessing a VSAM data set from multiple requests in other input and/or output processing modes.

Term	Description
SHAREOPTION 1	Any number of programs (TASKS, ACBs) could have access for input processing or only one program (TASK, ACB) can have access for output processing.
SHAREOPTION 2	Any number of programs (TASKS ACBs) could have access for input processing, and at the same time one program (TASK, ACB) can have access for output processing.
SHAREOPTION 3 & 4	Any number of programs (TASKS, ACBs) could have access to data set for both input and output processing. Note that data integrity is not ensured.

When Shareoption 1 or 2 is in effect for the data set in question, the following discussion is applicable. These considerations are given due to the fact that QUIKVSAM usage can involve multiple ddnames and/or alternate indexes referring to the same data set.

The same conditions would also apply when a data set has been opened by a TP network in a partition/region and QUIKVSAM is to access the same data set from a batch partition/region.

Item	Description
Fact	QUIKVSAM opens each ddname (ACB) for possible output processing on the condition the first CALL request is OPTION. Otherwise, it is opened for input processing only. You must use the OPTION CALL command for feedback communication whenever any ADD/UPDATE/ ERASE function is performed. You can use the OPT-RESET request if you want to load a VSAM file with the REUSE characteristics.

Item	Description	
Consider	To retrieve from a data set with multiple ddnames:	
	Solution 1	Do not use OPTION in a call command.
	Solution 2	Do a serve-no-purpose POINT or GET as the first call to QUIKVSAM then issue a CALL with OPTION command if feedback information is wanted.

Example

You have an employee master file you want to read sequentially from front to back. While doing so, you also want to read another record in random sequence from the same data set.

Solution with no OPTION feedback area:

```
CALL QUIKVSAM C'ddname1 ' C'GET'  WST1
CALL QUIKVSAM C'ddname2 ' C'READ' WST201 WST2
```

Solution with OPTION feedback area:

```
CALL QUIKVSAM C'ddname1 ' C'OPTION' WST1
CALL QUIKVSAM C'ddname2 ' C'OPTION' WST21
CALL QUIKVSAM C'ddname1 ' C'POINT'  WST1
CALL QUIKVSAM C'ddname2 ' C'POINT'  WST1
CALL QUIKVSAM C'ddname1 ' C'GET'    WST101
CALL QUIKVSAM C'ddname2 ' C'READ'   WST301 WST102
```

No adding/updating is allowed in this situation.

Functions

Examples and definitions of QUIKVSAM functions follow. The required and optional operands are defined. Optional operands are given in [brackets].

Add/Insert Sequential (KSDS)

ADD Command

Write randomly keyed logical records to the data set specified by ddname, the data located in the area designated by rec-area. The function inserts records into KSDS data sets based upon the unique key found in the record.

Record length must be specified in relative positions 4-7 (binary) of the options feedback area.

```
STMT              OP2         OP3     OP4
CALL QUIKVSAM     C'ddname'   C'ADD'  rec-area
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use ddname of CLOSE.
  - VISION:Report area containing the ddname.

  The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'ADD' in literal format.
  - VISION:Report area containing ADD.
- A VISION:Report area containing the record to be inserted into the data set by QUIKVSAM.

  The options feedback area should be used and the return/error codes (positions 8-9 of feedback area) should be checked for success or failure after each call to QUIKVSAM.

  Duplicate record= RC/EC = X'0808'

Close Data Set (KSDS, ESDS, RRDS)

CLOSE Command

The CLOSE function causes QUIKVSAM to close a particular data set specified by ddname or to close all currently open VSAM data sets.

```
STMT                  OP2          OP3
(A) CALL QUIKVSAM     C'ddname'    C'CLOSE'
(B) CALL QUIKVSAM     C'CLOSE'
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- For CALL format (A), the ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use a ddname of CLOSE.
  - VISION:Report area containing the ddname.

The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.

- For CALL format (B), the function/command can be one of the following:
  - C'CLOSE' in literal format.
  - VISION:Report area containing CLOSE.
- For CALL format (A), the function/command can be one of the following:
  - C'CLOSE' in literal format.
  - VISION:Report area containing CLOSE.

  With intervening CLOSE requests the same data set (ddname) can be accessed unlimited number of times in the same run.

  Also, by using the CLOSE command, unlimited number of data sets (ddnames) can be accessed, up to a maximum of eight for MVS, five for VSE, at any one time.

Close and Reopen Data Set (KSDS)

CLOSER Command

After loading a KSDS file, in order to reprocess it, the file must be closed. This VSAM requirement is for creating the index portion of the cluster and any alternate indexes associated with it. Then the file must be opened for subsequent processing. The CLOSER command accomplishes this in an efficient manner without releasing control blocks and regenerating them.

```
STMT              OP2          OP3
CALL QUIKVSAM     C'ddname'    C'CLOSER'
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use the ddname of CLOSE.
  - VISION:Report area containing the ddname.

  The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'CLOSER' in literal format.

- VISION:Report area containing CLOSER.

  With intervening CLOSER requests the same data set (ddname) can be accessed an unlimited number of times in the same run.

Erase Random and Sequential (KSDS,RRDS)

ERASE Command

This command erases the logical record most recently retrieved for update from the data set specified by ddname. The ERASE function deletes the logical record in the same mode in which it was retrieved (that is, sequential or random).

```
STMT                    OP2          OP3
CALL QUIKVSAM           C'ddname'    C'ERASE'
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use the CLOSE ddname.
  - VISION:Report area containing the ddname.

  The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'ERASE' in literal format.
  - VISION:Report area containing ERASE.

  The record location is not needed on an ERASE command. VSAM saves the identity of the record most recently retrieved for update (from this same data set), and this will be the record that is deleted.

  The options feedback area should be used and the return/error codes (positions 8-9 of feedback area) should be checked for success or failure after each call to QUIKVSAM.

Retrieve Sequential (KSDS,ESDS)

GET or SEQNTL Command

Retrieve the next sequential (or first) logical record from the data set specified by ddname into the area designated by rec-area.

```
STMT                OP2          OP3      OP4

CALL QUIKVSAM       C'ddname'    C'GET'   rec-area
                                 or
                                 C'SEQNTL'
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use the CLOSE ddname.
  - VISION:Report area containing the ddname.

  The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'GET' in literal format. Do not use the CLOSE ddname.
  - C'SEQNTL' in literal format.
  - VISION:Report area containing GET or SEQNTL.
- Record return area where QUIKVSAM places the retrieved record.

  This area contains high values when end of file is reached (RC/EC=X'0804'). The length of this area must be at least equal to the maximum record size as defined in the catalog for the data set specified. Sequential retrieval commences at the beginning of the file unless other requests for the same ddname have already occurred. After a successful point call is made, the record matching either the KEQ or KGE attributes is returned on a GET request.

Retrieve Sequential for Update (KSDS, ESDS)

GET-UPD Command

Retrieve for possible updating the next sequential (or first) logical record from the data set specified by ddname into the area designated by rec-area.

```
STMT              OP2         OP3          OP4
CALL QUIKVSAM     C'ddname'   C'GET-UPD'   rec-area
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use the CLOSE ddname.
  - VISION:Report area containing the ddname.

  The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'GET-UPD' in literal format.
  - VISION:Report area containing GET-UPD.
- Record return area where QUIKVSAM places the retrieved record. This area contains high values when end of file is reached (RC/EC = X'0804'). The length of this area must be at least equal to the maximum record size as defined in the catalog for the data set specified.

  A call to QUIKVSAM using OPTION must precede the first use of this command. The record retrieved with this command can be updated or erased on a KSDS file; whereas on an ESDS file, a retrieved record can only be updated.

  Sequential retrieval commences at the beginning of the file unless other requests for the same ddname have already occurred. After a successful point call is made, the record matching either the KEQ or KGE attributes is returned on a GET-UPD request.

Load/Insert Sequential (KSDS, ESDS)

LOAD Command

Write sequentially to the data set specified by ddname the logical record located in the area designated by rec-area.

In addition to the data set load/create capability, this function also inserts records (in key sequence) to an already existing KSDS data set or adds records at the logical end of an ESDS data set. Record length must be specified in relative positions 4-7 (binary) of the options feedback area.

```
STMT                OP2         OP3         OP4
CALL QUIKVSAM       C'ddname'   C'LOAD'     rec-area
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use the CLOSE ddname.
  - VISION:Report area containing the ddname.

  The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'LOAD' in literal format.
  - VISION:Report area containing LOAD.
- A VISION:Report area containing the record to be inserted into the data set by QUIKVSAM.

  A call to QUIKVSAM using OPTION must precede the first use of this command. As indicated above, record length must be specified in positions 4-7 of the feedback area.

  The options feedback area should be used and the return/error codes (positions 8-9 of feedback area) should be checked for success or failure after each call to QUIKVSAM.

  Duplicate record: RC/EC = X'0808'

  Record out of sequence: RC/EC = X'080C'

Set Up Communication/ Feedback Area (KSDS, ESDS, RRDS) Using OPEN

OPEN Command

The OPEN function allows you to designate an opt-area where the feedback value and return codes are placed for the data set specified by ddname. The password for the data set can also be specified.

This function communicates to QUIKVSAM that feedback data is wanted and where to place it for all subsequent calls made involving the data set specified by ddname. This command is read only and does not allow add/update/erase functions.

```
STMT                    OP2         OP3         OP4      OP5
CALL QUIKVSAM           C'ddname'   C'OPEN'     opt-area [C'password']
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use the CLOSE ddname.
  - VISION:Report area containing the ddname.

  The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'OPEN' in literal format.
  - VISION:Report area containing OPEN.
- A VISION:Report area where feedback values and return codes are placed after each call to QUIKVSAM for this same data set. The length of the option feedback area is 13 or 17 bytes depending on RBA or XRBA. See QUIKVSAM feedback area contents.
- Optional. This operand communicates to QUIKVSAM the password to be supplied when opening the VSAM data set specified in operand 2 as ddname. If a password is entered, it must be eight positions in length, padded with spaces, if necessary. This operand can be specified by one of the following methods: field definition, character literal, or hexadecimal literal.

  This operand is optional and is not required for data sets without password protection.

Any data (such as record length) that you want to place in the feedback area must be placed after an OPEN CALL request is made.

Set Up Communication/ Feedback Area (KSDS, ESDS, RRDS) Using OPTION and OPT-RESET

OPTION and OPT-RESET Command

The OPTION and OPT-RESET functions allow you to designate an opt-area where feedback values and return codes are placed for the data set specified by ddname. The password for the data set can also be specified. The OPTION function opens the VSAM file in update mode. The OPT-RESET function opens the VSAM file in load mode. OPT-RESET can be used when the VSAM file has the REUSE characteristics.

These two functions communicate to QUIKVSAM that feedback data is wanted and where to place it for all subsequent calls made involving the data set specified by ddname. An options feedback area is required by QUIKVSAM when functions requested are ADD/UPDATE/ERASE or LOAD if the function is OPT-RESET.

```
STMT                  OP2         OP3         OP4       OP5
CALL QUIKVSAM         C'ddname'   C'OPTION'   opt-area  [C'password']
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use the CLOSE ddname.
  - VISION:Report area containing the ddname.

  The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'OPTION' in literal format.
  - C'OPT-RESET' in literal format.
  - VISION:Report area containing OPTION or OPT-RESET.
- A VISION:Report area where feedback values and return codes are placed after each call to QUIKVSAM for this same data set. The length of the option feedback area is 13 or 17 bytes depending on RBA or XRBA. See QUIKVSAM feedback area contents.

- Optional. This operand communicates to QUIKVSAM the password to be supplied when opening the VSAM data set specified in operand 2 as ddname. If a password is entered, it must be eight positions in length, padded with spaces, if necessary. This operand can be specified by one of the following methods: field definition, character literal, or hexadecimal literal.

  This operand is optional and is not required for data sets without password protection.

Any data (such as record length) that you want to place in the feedback area must be placed after an OPTION or OPT-RESET CALL request is made.

Point/Generic Position (KSDS, ESDS, RRDS)

POINT or PNT Command

Position the data set specified by ddname to begin subsequent sequential retrieval at the logical record whose generic (or full) key value is located in key-area.

```
STMT                  OP2         OP3          OP4          OP5

CALL QUIKVSAM         C'ddname'   C'POINT'     key-area     [C'KGEnn'
                                    or                          or
                                  C'PNT'                     C'KEQ']
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use ddname of CLOSE.
  - VISION:Report area containing the ddname.

  The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'POINT' in literal format.
  - C'PNT' in literal format.
  - VISION:Report area containing POINT or PNT.
- A VISION:Report area where the key of the record that is to be positioned is located. It must be the length of full key for KEQ (key equal) POINT. It could be partial key length if the generic POINT option KGE (key greater or equal) is used (KSDS only).

To POINT to an ESDS or RRDS data set, the key must be specified in a four-byte binary format. The key of an ESDS record is its RBA (relative byte address), whereas an RRDS key is the relative record number.

No partial key or generic point is allowed with ESDS and RRDS data set records.

- Optional. This operand determines the type of point wanted by the user, and it can be one of the following:
  - C'KEQ' to point at a record with a key equal.
  - C'KGEnn' to point at a generic record with a key greater or equal.

    The nn on KGE represents the partial key length to be used in the point and must be two digits in the range of 01-99.
  - A VISION:Report area containing one of the above.

    This operand is optional with the default being KGE using the full key length.

    The options feedback area should be used and the return/error codes (positions 8-9 of feedback area) should be checked for success or failure after each POINT operation.

    Making a request for sequential retrieval following a point failure causes an abnormal termination.

    No record found: RC/EC = X'0810'

Retrieve Random (KSDS, ESDS, RRDS)

READ or RANDOM Command

Randomly retrieve from the data set specified by ddname a logical record matching the key-area and return into the area designated by rec-area.

STMT	OP2	OP3	OP4	OP5	OP6
CALL QUIKVSAM	C'ddname'	C'READ' or C'RANDOM'	rec-area	key-area	[C'KEQ' or C'KGE']

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use the CLOSE ddname.
  - VISION:Report area containing the ddname.

  The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'READ' in literal format.
  - C'RANDOM' in literal format.
  - VISION:Report area containing READ or RANDOM.
- Record return area where QUIKVSAM places the retrieved record. This area contains high values on a no-record-found condition (RC/EC=X'0810'). The length of this area must be at least equal to the maximum record size as defined in the catalog for the data set specified.
- An area where the key of the record to be retrieved is located. The length of this area for KSDS data sets is equal to the key length padded with fill characters to the right, if necessary.

  To randomly retrieve records from ESDS or RRDS data sets, the key must be specified in a 4-byte binary format. The key of an ESDS record is its RBA (relative byte address), whereas an RRDS key is the relative record number of the record wanted.
- Optional. This operand determines the type of match between the key wanted and the data set and can be one of the following:
  - C'KEQ' if equal key comparison wanted.
  - C'KGE' if a greater or equal comparison wanted.
  - VISION:Report area containing KEQ or KGE.

This operand is optional with the default being KEQ when not specified. ESDS and RRDS key search matches must use KEQ.

Retrieve Random for Update (KSDS, ESDS, RRDS)

Command: READ-UPD

Randomly retrieve for possible updating from the data set specified by ddname a logical record matching the key-area and return into the area designated by rec-area.

STMT	OP2	OP3	OP4	OP5	OP6
CALL QUIKVSAM	C'ddname'	C'READ-UPD'	rec-area	key-area	[C'KEQ' or C'KGE']

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use the CLOSE ddname.
  - VISION:Report area containing the ddname.

    The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'READ-UPD' in literal format.
  - VISION:Report area containing READ-UPD.
- Record return area where QUIKVSAM places the retrieved record. This area contains high values on a no-record-found condition (RC/EC = X'0810'). The length of this area must be equal to or greater than the maximum record size as defined in the catalog for the data set specified.
- Location at which the key of each record retrieved is placed. The length of this area for KSDS data sets is equal to the key length padded with fill characters to the right, if necessary.

  To randomly retrieve records from ESDS or RRDS data sets, the key must be specified in a 4-byte binary format. The key of an ESDS record is its RBA (relative byte address), whereas an RRDS key is the relative record number of the record wanted.
- Optional. This operand determines the type of match between the key wanted and the data set and can be one of the following:

- C'KEQ' if equal key comparison wanted.
- C'KGE' if a greater or equal comparison wanted.
- VISION:Report area containing KEQ or KGE.

Note: A call to QUIKVSAM using OPTION must precede the first use of this command.

This operand is optional with the default being KEQ when not specified. ESDS and RRDS key search matches must use KEQ.

The record retrieved with this command can be updated or erased on an ESDS or RRDS data set. On an ESDS data set, a record can only be updated. On a KSDS record, its length can be changed.

Add/Insert Random (RRDS)

RRADD Command

Write randomly relative-keyed logical records to the data set specified by ddname the contents located in the area designated by rec-area.

The key (relative record number) of the record must be placed in the key-area location. The function inserts records into RRDS data sets based upon the unique key of the record. Record length must be specified in relative positions 4-7 (binary) of the options feedback area.

```
STMT                 OP2          OP3         OP4        OP5
CALL QUIKVSAM        C'ddname'    C'RRADD'    rec-area   key-area
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use ddname of CLOSE.
  - VISION:Report area containing the ddname.

  The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'RRADD' in literal format.
  - VISION:Report area containing RADD.
- A VISION:Report area containing the record to be inserted in the data set by QUIKVSAM.

- Area where the key (relative record number) of the record to be inserted is located. The key of an RRDS record is in a 4-byte binary format.

  The options feedback area should be used and the return/error codes (positions 8-9 of feedback area) should be checked for success or failure after each call to QUIKVSAM.

  Duplicate record= RC/EC = X'0808'

Retrieve Sequential (RRDS)

RRGET Command

Retrieve the next sequential (or first) logical record from the data set specified by ddname into the area designated by rec-area. The key (relative record number) of the retrieved record is placed into key-area location.

STMT	OP2	OP3	OP4	OP5
CALL QUIKVSAM	C'ddname'	C'RRGET'	rec-area	key-area

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use the CLOSE ddname.
  - VISION:Report area containing the ddname.

  The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'RRGET' in literal format.
  - VISION:Report area containing RRGET.
- Record return area where QUIKVSAM places the retrieved record. This area contains high values when end of file is reached (RC/EC = X'0804').
- Location at which the key of each record retrieved is placed. The key is returned to this area which is in a 4-byte binary format.

  Sequential retrieval commences at the beginning of the file unless other requests for the same ddname have already occurred.

  After a successful point call is made, the record matching either the KEQ or KGE attributes are returned on a RRGET request.

Retrieve Sequential for Update (RRDS)

RRGET-UPD Command

Retrieve for possible update the next sequential (or first) logical record from the data set specified by ddname into the area designated by rec-area. The key (relative record number) of the retrieved record is placed into key-area location.

```
STMT                OP2          OP3           OP4          OP5
CALL QUIKVSAM       C'ddname'    'RRGET-UPD'   rec-area     key-area
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use the CLOSE ddname.
  - VISION:Report area containing the ddname.

  The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'RRGET-UPD' in literal format.
  - VISION:Report area containing RRGET-UPD.
- Record return area where QUIKVSAM places the retrieved record. This area contains high values when end of file is reached (RC/EC = X'0804').
- Location at which the key of each retrieved record is placed. The key is returned to this area which is in a 4-byte binary format.

  The record retrieved with this command can be updated or erased.

  Sequential retrieval commences at the beginning of the file unless other requests for the same ddname have already occurred.

  After a successful point call is made, the record matching either the KEQ or KGE attributes is returned on a RRGET-UPD request.

Load/Insert Sequential (RRDS)

RRLOAD Command

Note: A call to QUIKVSAM using OPTION must precede the first use of this command.

Write sequentially to the data set specified by ddname the logical record located in the area designated by rec-area.

The record will be written into the next available slot, and the key of that slot (relative record number) will be returned to the key-area location.

```
STMT                 OP2           OP3         OP4        OP5
CALL QUIKVSAM        C'ddname'     C'RRLOAD'   rec-area   key-area
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use the CLOSE ddname.
  - VISION:Report area containing the ddname.

  Note: A call to QUIKVSAM using OPTION must precede the first use of this command.

  The length of the ddname Operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'RRLOAD' in literal format.
  - VISION:Report area containing RRLOAD.
- A VISION:Report area containing the record to be inserted into the data set by QUIKVSAM.
- Location where the key of each record written is placed. The key is returned to this area which is in a 4-byte binary format.

  The record length must be placed into positions 4-7 of the feedback area in binary format.

  The options feedback area should be used and the return/error codes (positions 8-9 of feedback area) should be checked for success or failure after each call to QUIKVSAM.

Temporary Close (KSDS, ESDS, RRDS)

TCLOSE Command

A file can be closed temporarily and it will not be disconnected from processing; you can continue processing without opening the file (to do so is an error). The last block (control interval) accessed is written to the data set and, where applicable, index, alternate index update can occur (in essence, flushes VSAM buffers). You are responsible for positioning the file after TCLOSE and subsequent processing. An example of this is not repositioning after reaching EOF and executing the TCLOSE command.

```
STMT                    OP2             OP3
CALL QUIKVSAM           C'ddname'       C'TCLOSE'
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use the CLOSE ddname.
  - VISION:Report area containing the ddname.

The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.

- The function/command can be one of the following:
  - C'TCLOSE' in literal format.
  - VISION:Report area containing TCLOSE.

TCLOSE requests can access the same data set (ddname) an unlimited number of times in the same run.

Update/Change Random or Sequential (KSDS, ESDS, RRDS)

Command: UPDATE

Rewrite the logical record to the data set specified by ddname from the area designated by rec-area. The record must have been the most recent one retrieved for update from the same data set. Record lengths must be specified in relative positions 4-7 (binary) of the options feedback area. The update function will write the record to the data set in the same mode as it was retrieved (that is, sequential or random).

```
STMT                OP2           OP3             OP4
CALL QUIKVSAM       C'ddname'     C'UPDATE'       rec-area
```

Operands

- QUIKVSAM: Name of the VSAM subroutine.
- The ddname/file name can be one of the following:
  - C'ddname' in literal format. Do not use the CLOSE ddname.
  - VISION:Report area containing the ddname.

  The length of the ddname operand is system dependent (VSE—seven positions, MVS—eight positions) with the necessary spaces included for padding.
- The function/command can be one of the following:
  - C'UPDATE' in literal format.
  - VISION:Report area containing UPDATE.
- A VISION:Report area containing the record to be written to the data set by QUIKVSAM. If a record's length will not change, the correct length is already present in the feedback positions 4-7 from the retrieve for update. The length of a record can never exceed the maximum record size as defined in the catalog for the data set specified. VSAM does not allow changing the length of a record in an ESDS data set, and the records are always of a fixed length on a RRDS data set. You cannot change the key portion in the record of a KSDS data set.

  The options feedback area should be used and the return/error codes (positions 8-9 of feedback area) should be checked for success or failure after each call to QUIKVSAM.

Examples

This section shows examples using QUIKVSAM. The JCL for VSE and MVS is included with each example. VISION:Report statements are presented to demonstrate the use of QUIKVSAM. The appropriate load libraries or phase libraries are assumed to be included and are not shown. Even though AMS statements are MVS, most of the statements are applicable to VSE. It is assumed that VSE I/O statements, if required, will be added to the VISION:Report statements.

Example 1 — Define (using AMS) and Load a Variable Length Record VSAM Data Set

This example consists of two steps:

1 Define the VSAM data set into the catalog using Access Method Services.

2 Load the same data set using VISION:Report/QUIKVSAM.

This example applies to KSDS and ESDS data sets (without keys) with variable length records.

AMS Statements

```
DEFINE   CLUSTER     (NAME (VSAM.CLUSTER)     VOL (VOLSER)-
         FILE (DATASET)     RECSZ (80,300)     KEY (4,5)-
         CYL (10,1)        FREESPACE (20,5) )
/*
```

VISION:Report Statements

```
010 CALL QUIKVSAM C'DATASET ' C'OPTION' WST1 /* Setup feedback.
050 ATEND 300                               /* At EOF, GOTO 300.
100 GET                                     /* Get a record.
110 MOVE INF1-2-B TO WST21-25-P             /* MOVE rl to WST.
120 SUB C'4' FR WST21-25-P                  /* Subtract RDW Length
130 MOVE WST21-25-P TO WST4-7-B             /* MOVE to feedback rec. len.
150 CALL QUIKVSAM C'DATASET' C'LOAD' INF5 /* Put a record.
160 IF WST8-9 IS LOVALUE                    /* Check RC/EC.
170      GOTO 100.                          /* Zeroes, GOTO GET.
200 PRINTHEX INF1 INF1-2                    /* Prt hex input record
210 PRINTHEX WST1-13                        /* Prt hex VSAM feedback area
300 CALL QUIKVSAM C'CLOSE'                  /* Request CLOSE.
310 GOTO EOJ                                /* EOJ.
400 END
/*
```

If this example's AMS statements included the REUSE option, statement 010 could specify C'OPT-RESET' rather than C'OPTION'.

VSE JCL Example

```
//  JOB SAMPLE1 LOAD VAR-LEN RECORD FILE
//  ASSGN SYS010,251 ASSIGN DISK OUTPUT
//  DLBL DATASET,'VSAM.CLUSTER',,VSAM
//  EXTENT SYS010,VOLSER
//  ASSGN SYS011,180 ASSIGN TAPE INPUT
//  TLBL INF,'VSAM.BACKUP'
//  EXEC IDCAMS,SIZE=AUTO
. . . AMS STATEMENTS
//  EXEC QUKBJOB
INFTAPV50000304SSYS011
. . . .VISION:REPORT statements
```

MVS JCL Example

```
//SAMPLE1 JOB (800-0000,000),RK LOAD VAR-LEN RECORD DATA SET
//S1 EXEC  PGM=IDCAMS
//SYSPRINT DD SYSOUT=A
//DATASET  DD DISP=OLD,UNIT=3380,DSN=VSAM.CLUSTER
//SYSIN    DD *
. . .AMS STATEMENTS
//S2 EXEC  PGM=QUIKJOB
//SYSPRINT DD SYSOUT=A
//DATASET  DD DSN=VSAM.CLUSTER,DISP=OLD
//SYSUT1   DD DSN=VSAM.BACKUP,DISP=OLD,UNIT=TAPE
//SYSIN    DD *
. . . VISION:REPORT statements
```

Example 2 — Copy a VSAM Data Set to Tape

The example shown below illustrates how to copy a VSAM data set to tape in variable length format. This example could apply to KSDS and ESDS data sets with fixed, variable, or spanned length records.

```
010 CALL QUIKVSAM C'DATASET ' C'OPTION' WST1   /* Setup feedback.
020 CALL QUIKVSAM C'DATASET ' C'GET' OFA5      /* Get a record.
050 IF OFA5-8 IS HIVALUE                       /* Test EOF.
060      GOTO 950.
070 IF WST8-9 IS NOT LOVALUE                   /* Test RC/EC.
080      GOTO 930.
100 MOVE LOVALUE TO OFA1-4                     /* Zero rlxx OFA1-4.
110 MOVE WST6-7-B TO WST21-25-P                /* MOVE rl to WST.
120 ADD C'4' TO WST21-25-P                     /* Add 4 to rl.
130 MOVE WST21-25-P TO OFA1-2-B                /* MOVE rl into OFA1-2.
150 WRITE OFA                                  /* Put a record.
160 GOTO 020                                   /* Go do it again.
930 PRINTHEX WST1-13                           /* Prt hex (variable) areas
940 PRINTHEX OFA1 WST6-7                       /* On abnormal RC/EC.
950 CALL QUIKVSAM C'CLOSE'                     /* CLOSE.
960 GOTO EOJ                                   /* Force EOJ.
990 END
/*
```

Line 010 above opens VSAM file in update mode, although it is not required for this example.

VSE JCL Example

```
//  JOB SAMPLE2 COPY VSAM VAR-LEN TO TAPE
//  ASSGN SYS010,251 ASSIGN INPUT IF REQUIRED
//  DLBL IJSYSUC,'VSAM.UCAT',,VSAM USER CATALOG (OPTIONAL)
//  EXTENT SYS010,VOLSER
//  DLBL DATASET,'VSAM.CLUSTER',,VSAM
//  EXTENT SYS010,VOLSER
//  ASSGN SYS011,180 ASSIGN OUTPUT IF REQUIRED
//  TLBL OFA,'VSAM.BACKUP'
//  EXEC QUKBJOB
OFATAPV50000304SSYS011
```

MVS JCL Example

```
//SAMPLE2 JOB (800-0000,000),RK COPY VSAM VAR-LEN TO TAPE
//S2 EXEC  PGM=QUIKJOB
//SYSPRINT DD SYSOUT=A
//DATASET  DD DSN=VSAM.CLUSTER,DISP=OLD
//SYSUT2   DD DSN=VSAM.BACKUP,DISP=(NEW,KEEP),UNIT=TAPE,
//      VOL=SER=VOLSER,DCB=(RECFM=VB,LRECL=304,BLKSIZE=5000)
//SYSIN    DD *
```

Example 3 — Load VSAM Data Set with Fixed Length Records

This example loads records of a fixed length (100 bytes) into the VSAM data set with the ddname=DATSET1 and file-id or data set name of DATASET.MASTER.

A count of the number of records loaded is recorded by the ACCUM statement on line 060 and will be printed at EOJ.

```
EQU OPT-AREA WST1
EQU REC-AREA INF1
EQU RC-EC WST8-9
005 HDR 1A 1 $IPLDAT$ QUIKVSAM DATA SET LOAD/CREATE
010 CALL QUIKVSAM C'DATSET1 ' C'OPTION' OPT-AREA  /* Setup fdbk area.
020 MOVE C'100' TO WST4-7-B                       /* MOVE 100 to fdbk for rec len.
040 ATEND 200                                     /* At EOF goto EOJ processing.
050 GET                                           /* GET a record.
060 ACCUM ONE IN A 5 BYTE CTA, ON BREAKS PRINT IN POS 001 0C       /* Add 1.
070 CALL QUIKVSAM C'DATSET1 ' C'LOAD' REC-AREA    /* Put record to VSAM.
080 IF RC-EC IS LOVALUE                           /* Ck fdbk codes.
090      GOTO 050.                                /* Zeroes ok.
095 * Abnormal feedback codes dump areas, CLOSE and EOJ occurs.
100 PRINTHEX INF1-100                             /* Prt-hex record area.
110 PRINTHEX WST1-13                              /* Dump feedback codes.
120 PRINTHEX CTA1-B                               /* Prt-hex rec ctr.
190 * EOJ Processing
200 CALL QUIKVSAM C'CLOSE'                        /* CALL to CLOSE data sets.
250 GOTO EOJ /* Force EOJ and rec count.
500 END
/*
```

VSE JCL Example

```
//  JOB SAMPLE3 QUIKVSAM DATA SET LOAD
//  ASSGN SYS010,280 ASSIGN INPUT IF REQUIRED
//  TLBL INF,'BACKUP'
//  ASSGN SYS016,152 ASSIGN OUTPUT IF REQUIRED
//  DLBL DATSET1,'DATASET.MASTER',,VSAM
//  EXTENT SYS016,VOLSER
//  EXEC QUKBJOB
INFTAPE40000100SSYS010
```

MVS JCL Example

```
//SAMPLE3 JOB (800-0000,000),NAME QUIKVSAM DATA SET LOAD
//S1 EXEC  PGM=QUIKJOB
//SYSPRINT DD SYSOUT=A
//DATSET1  DD DSN=DATASET.MASTER,DISP=OLD
//SYSUT1   DD DSN=BACKUP,DISP=OLD,UNIT=TAPE,VOL=SER=VOLSER
//SYSIN    DD *
```

Example 4 — Retrieve Records Sequentially for UPDATE or ERASE

The following statements show how to sequentially retrieve VSAM records for possible updating or deletion. A count of records read, updated, and erased is accumulated.

This example is applicable to KSDS and ESDS data sets with fixed or variable length records (ESDS without erase).

```
005 HDR 1A 1 $IPLDAT$    QUIKVSAM DATA SET UPDATE/ERASE
007 HDR 2A 0 RECORDS-READ     UPDATED             ERASED
010 CALL QUIKVSAM C'MASTER  ' C'OPTION' WST201        /* Setup feedback area.
090 *
100 CALL QUIKVSAM C'MASTER  ' C'GET-UPD' WST1         /* Get a record/for upd.
120 PERFORM 900 THRU 919                              /* Do feedback check.
130 ACCUM ONE IN A 5 BYTE CTA, ON BREAKS PRINT IN POS 001 0C
140 IF WST66-70-P IS ZERO                             /* Check if YTD $ is zero.
150      GOTO 300.                                    /* Yes - go delete it.
200 MOVE ZEROS TO WST66-70-P                          /* Not zero, zero fld.
220 CALL QUIKVSAM C'MASTER  '  C'UPDATE' WST1         /* Write back record.
230 PERFORM 900 THRU 919                              /* Do feedback check.
250 ACCUM ONE IN A 5 BYTE CTB, ON BREAKS PRINT IN POS 021 0C
260 GOTO 100
290 *
300 CALL QUIKVSAM C'MASTER  ' C'ERASE'                /* ERASE rec, inactive.
320 PERFORM 900 THRU 919                              /* Do feedback check.
330 ACCUM ONE IN A 5 BYTE CTC, ON BREAKS PRINT IN POS 041 0C
350 GOTO 100
890 * TEST FOR EOD/EOF, RC/EC FOR ABNORMAL CONDITIONS
900 IF WST1-5 IS HIVALUE                              /* Test for EOF.
905      GOTO 950.
910 IF WST208-209 IS NOT LOVALUE                      /* Check RC/EC codes.
915      GOTO 930.
919 EXIT                                              /* Return when all is well.
929 * Prt-hex active areas on abnormal RC/EC condition
930 PRINTHEX WST1 WST204-207                          /* Prt hex variable.
940 PRINTHEX WST201-213                               /* Display WST area.
949 * E-O-J processing
950 CALL QUIKVSAM C'CLOSE'                            /* Close all VSAM data sets.
960 GOTO EOJ                                          /* Force EOJ.
990 END
/*
```

VSE JCL Example

```
//  JOB SAMPLE4 QUIKVSAM UPDATE/ERASE
//  ASSGN SYS010,163    ASSIGN INPUT IF REQUIRED
//  DLBL MASTER,'APPL.MASTER',,VSAM
//  EXTENT SYS010,VOLSER
//  EXEC QUKBJOB
```

MVS JCL Example

```
//SAMPLE4 JOB (800-0000,000) BOB QUIKVSAM UPDATE/ERASE
//S1  EXEC PGM=QUIKJOB
//SYSPRINT DD SYSOUT=A
//MASTER   DD DSN=APPL.MASTER,DISP=OLD
//SYSIN    DD *
```

Example 5 — Random/Sequential Retrieve with UPDATE and ADD

The example below illustrates the use of two VSAM data sets. An ESDS defined cluster is being read sequentially (line 050), placing the record in an area beginning at CTA1.

A search key (CTA5) from the ESDS is used to retrieve records randomly from a KSDS (line 100). Records are retrieved into an area beginning at CTA401, and feedback values are placed in WST21-33.

Depending on whether or not a record was found matching the search key, an update (line 200) or add (line 300) request is made to QUIKVSAM.

When the end of data on the ESDS is recognized (line 900), control is transferred to line 950; a call is made to close the VSAM data sets.

```
010 CALL QUIKVSAM C'ESDS    ' C'OPTION' WST1          /* Setup fdbk areas.
020 CALL QUIKVSAM C'KSDS ' C'OPTION' WST21
050 CALL QUIKVSAM C'ESDS    ' C'GET' CTA1             /*Get a record.
060 PERFORM 900 THRU 919                              /*Ck EOF, RC/EC.
100 CALL QUIKVSAM C'KSDS   'C'READ-UPD' CTA401 CTA5   /*Read KSDS for upd.
120 IF CTA401-405 IS HIVALUE                          /*Test if record found.
130      GOTO 250.
140 PERFORM 920 THRU 929                              /*Ck RC/EC.
150 MOVE CTA21-30 TO CTA451                           /*Do update
160 ADD CTA41-48 TO CTA461-466-P                      /*processing.
200 CALL QUIKVSAM C'KSDS    ' C'UPDATE' CTA401        /*Rewrite KSDS.
210 PERFORM 920 THRU 929                              /*Ck RC/EC.
220 GOTO 050
250 MOVE SPACES TO CTA401-800                         /*Clear area.

260 MOVE WST4-7 TO WST24-27                           /*Put reclen in feedback.
270 CALL QUIKMOVE CTA1 CTA401 WST6                    /*MOVE var length rec.
300 CALL QUIKVSAM C'KSDS    ' C'ADD' CTA401           /*Write record.
310 PERFORM 920 THRU 929                              /*Ck RC/EC.
320 GOTO 050
900 IF CTA1-5 IS HIVALUE                              /*Test for EOF.
905      GOTO 950.
910 IF WST8-9 IS NOT LOVALUE                         /*Test for abnormal conditions.
915      GOTO 930.
919 EXIT/*Ok, return.

920 IF WST28-29 IS NOT LOVALUE                        /* Test for abnormal.
925      GOTO 930.
929 EXIT                                              /* Ok, return.
930 PRINTHEX WST1-40                                  /* Prt hex feedback.
940 PRINTHEX CTA1-800                                 /* Prt hex records.
950 CALL QUIKVSAM C'CLOSE'                            /* CLOSE VSAM files.
960 GOTO EOJ                                          /* Force EOJ.
990 END
/*
```

VSE JCL Example

```
//  JOB SAMPLE5  ESDS/KSDS GET/UPDATE/ADD
//  ASSGN SYS008,152  ASSIGN INPUT IF REQUIRED
//  DLBL ESDS,'TRANS.DATA',,VSAM
//  EXTENT SYS008,VOLSER
//  ASSGN SYS009,153  ASSIGN INPUT IF REQUIRED
//  DLBL KSDS,'KSDS.MASTER',,VSAM
//  EXTENT SYS009,VOLSER
//  EXEC QUKBJOB
```

MVS JCL Example

```
//SAMPLE5  JOB (800-0000,000)      ESDS/KSDS GET/UPDATE/ADD
//S1  EXEC PGM=QUIKJOB
//SYSPRINT DD SYSOUT=A
//ESDS     DD DSN=TRANS.DATA,DISP=OLD
//KSDS     DD DSN=KSDS.MASTER,DISP=OLD
//SYSIN    DD *
```

Example 6 — Point and Sequential Retrieval

The succeeding example uses a data key found at INF1 to position (point) the data set for sequential retrieval beginning with the record matching the key.

The point call (line 100) shown uses the KEQ (key equal) option to search for a record matching the search key argument. The default on a point call is KGE (key greater or equal).

The return/error codes are tested for a successful point operation (line 110). Records are read from the data set (line 200) until a change in group occurs.

```
010 CALL QUIKVSAM C'DATSET2 ' C'OPTION' WST101            /* Setup fdbk area.
040 ATEND 950
050 GET /* GET trigger ord.#.
100 CALL QUIKVSAM C'DATSET2 ' C'POINT' INF1 C'KEQ'        /* Pos. to ord.#.
110 IF WST108-109 IS LOVALUE                              /* Check RC/EC.
120      GOTO 200.
140 MOVE INF1-20 TO PRT1                                  /* Indicate
150 MOVE C'ORDER NOT FOUND' TO PRT31                      /* not found.
160 PRINT
170 GOTO 050
200 CALL QUIKVSAM C'DATSET2 ' C'GET' WST1                 /* Get a VSAM rec.
210 IF WST108-109 IS NOT LOVALUE                          /* Check RC/EC.
220      GOTO 930.
230 IF WST1-10 IS NOT EQ TO INF1-10                       /* Check for end of grp.
240      GOTO 050.
250 MOVE WST1-100 TO PRT1                                 /* Print ord rec.
260 PRINT
270 GOTO 200                                              /* GET next.
930 PRINTHEX INF1-20                                      /* Prt hex ord,
940 PRINTHEX WST1-120                                     /* RC/EC and rec area.
950 CALL QUIKVSAM C'CLOSE'                                /* CLOSE file.
960 GOTO EOJ.                                             /* Force EOJ.
990 END
/*
```

Line 010 above opens VSAM file in update mode.

VSE JCL Example

```
//  JOB SAMPLE6    VSAM POINT/SEQ
//  ASSGN SYS010,161   ASSIGN INPUT IF REQUIRED
//  DLBL DATSET2,'OPEN.ORDERS',,VSAM
//  EXTENT SYS010,VOLSER
//  EXEC QUKBJOB
INFCARD
```

MVS JCL Example

```
//SAMPLE6  JOB (800-0000,000),RON VSAM POINT/SEQ
//S1  EXEC PGM=QUIKJOB
//SYSPRINT DD SYSOUT=A
//DATSET2  DD DSN=OPEN.ORDERS,DISP=OLD
//SYSIN    DD *
```

Example 7 — Sequential Retrieval with BREAKS, Accumulative Reporting, and Dummy INF Input

A call is made to QUIKVSAM to retrieve records from a payroll data set into the INF area. This area is used for ACCUM reporting and BREAKs which must be done in the INF area.

In the VSE JCL example shown following the code, the corresponding definitions for the INF file are specified as IGN. In the MVS JCL example definitions are specified as DUMMY.

```
000 HDR 1A 1    $IPLDAT$               PAYROLL DEMO
001 HDR 1B                PAGE $PG$
002 HDR 2A 0                           DATE OF   DATE OF PREVIOUS
003 HDR 2B      HRLY       THIS        Y.T.D.
004 HDR 3APLT DEPT EMP. EMPLOYEE NAME .BIRTH..EMPLOYMENT EDUC YTD GROSS
005 HDR 3B WORK RATE WEEK GROSS STATE TAX YTD FICA CURR FICA
010 BREAK 1 INF3-5 SB 1 SA 1 PRINT C'DEPT TOTS' IN TOT POS 011
050 CALL QUIKVSAM C'PRMASTR ' C'GET' INF1
051 IF INF1-5 IS HIVALUE                                /* Test for EOF.
052      GOTO EOJ.
054 IF INF3-5 IS EQ TO C'005' OR
055      IF INF3-5 IS EQ TO C'010'
056      GO TO 060.
057 GO TO 050
060 CHECKBREAKS
120 MOVE INF1-2 TO PRT2                                 /* MOVE plant to print.
130 MOVE INF3-5 TO PRT6                                 /* MOVE dept to print.
150 MOVE INF6-9 TO PRT10 0                              /* MOVE emp nr and zero
suppress.
160 MOVE INF10-25 TO PRT15                              /* MOVE emp name to print.
170 MOVE INF26-27 TO PRT32                              /* MOVE date of birth.
171 MOVE INF28-29 TO PRT35
172 MOVE INF30-31 TO PRT38
174 MOVE INF32-33 TO PRT42                              /* MOVE date of employment.
175 MOVE INF36-37 TO PRT48
180 MOVE INF38-39 TO PRT52                              /* MOVE education.
185 MOVE INF54-57-P TO PRT55 2C                         /* MOVE previous YTD gross.

190 MOVE INF50-53 TO PRT66 2                            /* MOVE hours worked.
200 MOVE INF46-49 TO PRT72 3                            /* MOVE hourly rate.
210 MOVE INF50-53 2D BY INF46 49 3D GIVING WST1-4-P 2DR
    /* Extend hrs wkd.
230 MOVE WST1-4-P TO PRT78 2C                           /* This weeks gross.
240 MOVE INF58-51-B TO WST 5-8-P

                                    /* Binary YTD  state tax., cvt to pkd dec.
250 MOVE WST5-8-P TO PRT89 2C                          /* Edit YTD st tax to print.
260 MOVE INF62-66 TO PRT102 2C                         /* MOVE YTD fica to print.
261 MULT WST1-4-P 2D BY C'058' 3D GIVING WST9-11-P 2DR  /* Calc current FICA.
265 MOVE WST9-11-P TO PRT113 2                         /* MOVE current FICA to print.
268 PRINT                                              /* PRINT the detail line.
300 ACCUM INF54-57-P IN A 5 BYTE CTR ON BREAKS PRINT IN POS 052 2C
310 ACCUM WST1-4-P  IN A 4 BYTE CTR ON BREAKS PRINT IN POS 078 2C
320 ACCUM WST9-11-P IN A 4 BYTE CTR ON BREAKS PRINT IN POS 110 2C
330 GO TO 060
999 END
/*
```

VSE JCL Example

```
//  JOB SAMPLE7       QUIKVSAM W/ACCUM
//  ASSGN SYS010,IGN     ASSIGN INF IGNORE
//  ASSGN SYS012,134  ASSIGN INPUT IF REQUIRED
//  DLBL PRMSTR,'VSAM.PAYROLL.DEMO',,VSAM
//  EXTENT SYS012,VOLSER
//  EXEC QUKBJOB
INFTAPE01000100SSYS010
```

MVS JCL Example

```
//SAMPLE7 JOB (800-0000,000),JIM QUIKVSAM W/ACCUM
//S1 EXEC  PGM=QUIKJOB
//SYSPRINT DD SYSOUT=A
//PRMASTR  DD DSN=VSAM.PAYROLL.DEMO,DISP=OLD
//SYSUT1   DD DUMMY,DCB=(RECFM=F,LRECL=100,BLKSIZE=100)
//SYSIN    DD *
```

Appendix

A

Invoking VISION:Report from VISION:Results or VISION:Eighty

From your VISION:Results (formerly known as DYL-280 II) or VISION:Eighty program (formerly known as DYL-280), you can invoke stand-alone VISION:Report applications and VISION:Report applications that access IMS or CA-IDMS/DB data bases. The following pages contain three examples of JCL statements and VISION:Report programs that illustrate how to accomplish this.

- MVS and VSE JCL invoking VISION:Report from VISION:Results or VISION:Eighty (see MVS JCL through Basic VISION:Report Program).
- MVS JCL for invoking VISION:Report accessing CA-IDMS/DB (see VISION:Report MVS JCL with CA-IDMS/DB and A Typical VISION:Report Program to Call CA-IDMS/DB Functions).
- MVS JCL for invoking VISION:Report accessing IMS (see VISION:Report MVS JCL with IMS and A Typical VISION:Report Program to Call IMS Functions).

Each example consists of two parts: the JCL statements and the VISION:Report program. The VSE JCL required for these jobs would be the equivalent of the MVS JCL. All three examples have the following in common:

- The size of the REGION in the STEP01 statement is user-defined because it depends on the size of the program and what other applications it may be accessing
- Wherever the word “user” or “your” appears in lowercase in the JCL statements, a user-defined file name (if applicable) should be entered.
- The SYS004 statement in the JCL must be coded exactly in the format shown in the examples.
- The VISION:Report program must begin with the VISION:Results or VISION:Eighty statement OPTION QUIKJOB.

DYLQKIMS

The VISION:Results or VISION:Eighty program executed in the MVS JCL with CA-IDMS/DB is DYLQIDMS; the MVS JCL with IMS is DYLQKIMS. Note that the installation procedures and the use of these two programs are the same as the VISION:Report programs QUIKIDMS and QUIKIMS as outlined in Chapter 6, Optional Material. (See the examples—STEP01 statements—on VISION:Report MVS JCL with CA-IDMS/DB and VISION:Report MVS JCL with IMS.)

To run VISION:Report from a VISION:Results or VISION:Eighty program, you must be on VISION:Results Release 3.0 or later or VISION:Eighty Release 6.0 or later.

MVS JCL

```
//JOB  (your standard job statement)
//STEP01  EXEC PGM=DYL280,REGION=768K

//STEPLIB   DD DSN=your.DYL280.LOADLIB,          VISION:Results Release 3.0
//             DISP=SHR                          or later or VISION:Eighty
                                                 Release 6.0 or later
                                                 load library

//          DD DSN=your.QUIKJOB.MVS.LOADLIB,     VISION:Report
//             DISP=SHR                          load library
//SYSPRINT  DD SYSOUT=*
//SYS280R   DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYS004    DD UNIT=ISPDA,SPACE=(TRK,(5))        required format when
                                                 calling VISION:Report
//SORTLIB   DD DSN=SYS1.SORTLIB,DISP=SHR         required
//SORTWK01  DD UNIT=SYSDA,SPACE=(TRK,(20,20))    only if
//SORTWK02  DD UNIT=SYSDA,SPACE=(TRK,(20,20))    SORT is used
//SORTWK03  DD UNIT=SYSDA,SPACE=(TRK,(20,20))
//SYSDET    DD *                                 for VISION:Report
                                                 card input only
                                                 (if applicable)

//SYSIN     DD *
    OPTION QUIKJOB
     .
     .  VISION:Report program
     .
```

VSE JCL

```
// JOB OPTIONQJ
// DLBL DYL282,'DYL280II.SP30.LIBS'  VISION:EIGHTY PHASE LIB
// EXTENT ,DOS001
// DLBL QIKCIL,'QUIKJOB.LIB'  VISION:REPORT PHASE LIB
// EXTENT ,DOSLB1
// LIBDEF PHASE,SEARCH=(DYL282.PHASE,QIKCIL.PHASE)
// ASSGN SYS004,DISK,VOL=DOSWRK,SHR
// DLBL IJSYS04,'==WORK1',0                        required
// EXTENT SYS004.DOSWRK,1,0,1,10
// EXEC DYL280,SIZE=768K
    OPTION QUIKJOB
     .
     .  VISION:Report program
     .
/*
/&
```

Basic VISION:Report Program

OPTION QUIKJOB is a VISION:Results or VISION:Eighty statement that must be present in order to call VISION:Report from VISION:Results or VISION:Eighty.

The OPTION SEQCHK=NO statement is the VISION:Report option, and SEQCHK=NO is only one of the operands available. If no VISION:Report option statement is entered, the default values are used.

```
OPTION QUIKJOB
OPTION SEQCHK=NO
     1000   GET  DET ATEND EOJ
            MOVE  DET1-80 TO PRT1
            PRINT
            GO TO 1000
            END

ENTER INPUT DATA RECORDS FOR SYSDET FILE HERE.
```

VISION:Report MVS JCL with CA-IDMS/DB

```
//JOB  (your standard job statement)
//STEP01  EXEC PGM=DYLQIDMS,REGION=2500K
//STEPLIB   DD DSN=your.QUIKJOB.MVS.LOADLIB,      VISION:Report
//          DISP=SHR                              load library

//          DD DSN=your.DYL280.LOADLIB,           VISION:Results Release 3.0
            DISP-SHR                              or later or VISION:Eighty
                                                  Release 6.0 or later
                                                  load library
//          DD DSN=IDMS.PROD10.LOADLIB,           CA-IDMS/DB
//          DISP=SHR                              load library
//          DD DSN=user.QUIK.LOAD,DISP=SHR        user's application
                                                  program library
//SYSCOPY   DD DSN=user.DYLIDMS.MVS,              user's copy
//          DISP=SHR                              library
//SYSPRINT  DD SYSOUT=*
//SYS280R   DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYS004    DD UNIT=ISPDA,SPACE=(TRK,(5))         required format when
                                                  calling VISION:Report
//SORTLIB   DD DSN=SYS1.SORTLIB,DISP=SHR          required only
//SORTWK01  DD UNIT=SYSDA,SPACE=(TRK,(20,20))     if SORT is used
//SORTWK02  DD UNIT=SYSDA,SPACE=(TRK,(20,20))
//SORTWK03  DD UNIT=SYSDA,SPACE=(TRK,(20,20))
//SYSJRNL   DD DUMMY
//J1JRNL    DD DUMMY
//J2JRNL    DD DUMMY
//J3JRNL    DD DUMMY
//J4J4NL    DD DUMMY
//DICTDB    DD DSN=IDMS.PROD10.DICTDB,            CA-IDMS/DB
//             DISP=SHR                           dictionary
//DYIDMSIN  DD DSN=user.QUIK.LOAD,DISP=SHR        user's VISION:Report
                                                  application library
//ORGDEMO   DD DSN=IDMS.PROD10.ORGDEMO,           CA-IDMS/DB
//             DISP=SHR                           data base
//SYSDET    DD *                                  for VISION:Report
                                                  card input only
                                                  (if applicable)
//SYSIN     DD *
```

A Typical VISION:Report Program to Call CA-IDMS/DB Functions

OPTION QUIKJOB is a VISION:Results or VISION:Eighty statement that must be present in order to call VISION:Report from VISION:Results or VISION:Eighty.

The OPTION SEQCHK=NO statement is the VISION:Report option, and SEQCHK=NO is only one of the operands available. If no VISION:Report option statement is entered, the default values are used.

Note: For more information on DYLQKIMS see DYLQKIMS.

```
OPTION QUIKJOB
OPTION SEQCHK=NO
EQU BIND-RUN-UNIT       PCB155
EQU BIND-RN             PCB144
EQU READY-RETRIEVE      PCB133
EQU OBTN-1ST-RN-AN      PCB115
EQU OBTN-NXT-RP=AN      PCB107
EQU FINISH              PCB98
EQU OBTAIN-SUF          PCB139
EQU COMM-BLOCK          PCB1-216
EQU PGM-NAME            PCB1-8
EQU ERR-STAT            PCB9-12
EQU DBKEY               PCB13-16-B
EQU DIR-DBKEY           PCB197-200-B
EQU REC-OCCUR           PCB209-212-B
EQU DML-SEQ             PCB213-216-B
EQU OFFICE-RN           SAV1-16
EQU OFFICE-AN           SAV17-32
EQU OFFICE              WST1-90
EQU OFFICE-STREET       WST1-20
EQU OFFICE-CITY         WST21-35
EQU OFFICE-STATE        WST36-37
EQU OFFICE-ZIP          WST38-46
EQU OFFICE-CODE         WST47-49
EQU EMPNO               WST50-55
EQU EMN                 WST56-56
EQU FNAME               WST57-68
EQU FNB                 WST69-69
EQU LNAME               WST70-84
EQU MMMMMM              WST85-85
EQU SALARY              WST86-90

HDR 1A 1 DYL CALLING QUIKIDMS
HDR 1B $IPLDAT$ PAGE $PG$
(Computer display)8    GET   DET ATEND   10
      MOVE  DET1-80 TO PRT1
      PRINT
      GOTO 8

10    MOVE   SPACE               TO SAV1-100
      MOVE   C'QUIKIDMS'         TO PGM-NAME
      MOVE   ZEROES              TO DBKEY
      MOVE   ZEROES              TO DIR-DBKEY
      MOVE   ZEROES              TO REC-OCCUR
      MOVE   C'1'                TO DML-SEQ
```

```
      MOVE   C'OFFICE'           TO OFFICE-RN
      MOVE   C'ORG-DEMO-REGION'  TO OFFICE AN

20    CALL QJBIDMS BIND-RUN-UNIT COMM-BLOCK
               C'EMPSS01'
25    CALL QJBIDMS BIND-RN  OFFICE-RN OFFICE
30    CALL QJBIDMS READY-RETRIEVE
35    PERFORM 900 THRU 990
40    CALL QJBIDMS OBTN-1ST-RN-AN  OFFICE-RN
               OFFICE-AN  OBTAIN-SUF
      GOTO 60
50    CALL QJBIDMS OBTN-NXT-RN-AN  OFFICE-RN
               OFFICE-AN OBTAIN-SUF
60    IF ERR-STAT IS EQ TO C'0307'
           GOTO 200.
70    PERFORM 900 THRU 990
90    MOVE OFFICE-STREET          TO PRT1-20
      MOVE OFFICE-CITY            TO PRT23-37
      MOVE OFFICE-STATE           TO PRT40-41
      MOVE OFFICE-ZIP             TO PRT44-52
      MOVE OFFICE-CODE            TO PRT55-57
      MOVE EMPNO                  TO PRT60-65
      MOVE EMN                    TO PRT68-68
      MOVE FNAME                  TO PRT71-82
      MOVE FNB                    TO PRT85-85
      MOVE LNAME                  TO PRT88-102
      MOVE MMMMMM                 TO PRT105-105
      MOVE SALARY                 TO PRT107-112
      PRINT
100   GOTO 50

200   CALL QJBIDMS FINISH
      GOTO EOJ

900   IF ERR-STAT IS EQ TO C'0000'
           GOTO 990.
      MOVE C'QUIKIDMS ABNORMAL STATUS' TO PRT1
      PRINT DOUBLESPACE
      PRINTHEX COMM-BLOCK
      PRINTHEX OFFICE
      GOTO EOJ

990   EXIT
      END


ENTER INPUT DATA RECORDS FOR SYSDET FILE HERE.
```

VISION:Report MVS JCL with IMS

```
//JOB  (your standard job statement)
//STEP01  EXEC PGM=DFSRRC00,REGION=4500K,
//             PARM='DLI,DYLQKIMS,PSBQREAD'
//STEPLIB   DD DSN=your.DYL280.LOADLIB,          VISION:Results Release 3.0
             DISP=SHR                            or later or VISION:Eighty
                                                 Release 6.0 or later
                                                 load library
//          DD DSN=your.QUIKJOB.MVS.LOADLIB,     VISION:Report
//             DISP=SHR                          load library
//          DD DSN=IMSVS.RESLIB,DISP=SHR         IMS load
//IMS       DD user.IMS.LOAD,DISP=SHR            library
//DBDLIB    DD DSN=user.IMS.LOAD,DISP=SHR        user's DBDLIB
//PSBLIB    DD DSN=user.IMS.LOAD,DISP=SHR        user's PSBLIB
//SYSCOPY   DD DSN=user.IMS.LOAD,DISP=SHR        user's COPYLIB
//SYSPRINT  DD SYSOUT=*
//ABNLIGNR  DD DUMMY
//SYSUDUMP  DD SYSOUT=*
//SYS280R   DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYS004    DD UNIT=ISPDA,SPACE=(TRK(5))         required format when
                                                 calling VISION:Report
//SORTLIB   DD DSN=SYS1.SORTLIB,DISP=SHR         required
//SORTWK01  DD UNIT=SYSDA.SPACE=(TRK,(20,20))    only if
//SORTWK02  DD UNIT=SYSDA,SPACE=(TRK,(20,20))    SORT is used
//SORTWK03  DD UNIT=SYSDA,SPACE=(TRK,(20,20))
//DFSRESLB  DD DSN=IMSVS.RESLIB,DISP=SHR         IBM IMS
                                                 library
//DFSVSAMP  DD DSN=user.xx.DFSVSAMP(xxxxxx),     buffer
//             DISP=SHR                          information
//IEFRDER   DD DUMMY                             for IMS
//ARMFIDX  DD DSN=user.VSAM.ARMFIDX,DISP=SHR     IMS
//ARMFIMS  DD DSN=user.VSAM.ARMFIMS,DISP=SHR     data base
//SYSDET   DD *                                  for VISION:Report
                                                 card input only
                                                 if applicable
//SYSIN    DD *
```

A Typical VISION:Report Program to Call IMS Functions

The OPTION QUIKJOB statement is a VISION:Results or VISION:Eighty requirement and must be present in order for VISION:Report to run.

The OPTION SEQCHK=NO statement is the VISION:Report option, and SEQCHK=NO is only one of the operands available. If no VISION:Report option statement is entered, the default values are used. (WSTSIZE=5000 is required only for this sample program.)

Note: For more information on DYLQKIMS, see DYLQKIMS.

```
OPTION QUIKJOB
OPTION SEQCHK=NO,WSTSIZE=5000
EQU STATUS-CODE          PCB11-12
EQU COUNT                WST1-4 0C ZERO

1200  CALL QJBTDLI C'GN ' C'01' WST1000
      ADD C'1' TO COUNT
      IF STATUS-CODE EQ C'GB'
           GOTO 1600.
      IF COUNT GT C'0100'
           GOTO 1600.
      IF STATUS-CODE IS EQ TO C'GA'  OR
      IF STATUS-CODE IS EQ TO C'GK'
           GOTO 1500.
      IF STATUS-CODE IS NOT EQ TO C'  '
           GOTO 1600.
1500  IF COUNT GT C'0050'
           GOTO 1200.
1550  MOVE COUNT TO PRT5-8
      MOVE STATUS-CODE TO PRT14
      MOVE WST1000-1050 TO PRT20
      PRINT
      GOTO 1200.

1600  MOVE BLANK TO PRT1-80
      MOVE STATUS-CODE TO PRT30-31
      MOVE C'STATUS CODE' TO PRT15-25
      MOVE C'COUNT = ' TO PRT35-42
      MOVE COUNT TO PRT47-50
      PRINT
      READ INPUT DATA

2000  GET DET ATEND EOJ
      MOVE DET1-80 PRT1
      PRINT
      GO TO 2000.
      END

ENTER DATA INPUT RECORDS FOR SYSDET FILE HERE.
```

Index

#

$

3

4

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z