

Advantage™ VISION:Builder®

Advantage™ VISION:Two™ for

OS/390®

Environment Guide

14.0



Computer Associates™

BUENM140.PDF/D92-013-014

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

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

Notwithstanding the foregoing, the user may print a reasonable number of copies of this documentation for its own internal use, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software of the user will have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to return to CA the reproduced copies or to certify to CA that same have been destroyed.

To the extent permitted by applicable law, CA provides this documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will CA be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of this documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised of such loss or damage.

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

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013(c)(1)(ii) or applicable successor provisions.

© 2002 Computer Associates International, Inc. (CA).

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

Contents

Chapter 1: Introduction

Configuration Requirements	1-1
Organization and Standards for This Book	1-2
Overview of VISION:Builder Application Development	1-3
Definitions	1-5
Processing Data	1-6

Chapter 2: VISION:Builder Runs, Run Control, and Execution JCL

Relationship Between Run Control, VISION:Builder Processing, and JCL.....	2-1
Files and Their DDnames in a VISION:Builder Run	2-2
Job Control Statements	2-2
Run Control Group	2-5
The Definition Maintenance Run.....	2-8
The Application Run.....	2-9
Single-Step Process-Sort-Report Run	2-9
Three-Step Process-Sort-Report Run	2-11
Single-Step No-Sort Report Run	2-15
Run Control and Execution JCL for Sample VISION:Builder Applications.....	2-17
Alternate Report Files	2-18
Report from Master File and Coordinated Files (Three-Step)	2-22
Create a Subfile and Generate a Report on an Alternate List File (Three-Step)	2-26
Update a Master File (Single-Step Sort) ④.....	2-29
Scan/Sample Report.....	2-32
Program Analyzer	2-34
Report Summary File.....	2-42
Update-in-Place (Single-Step No-Sort)④	2-48

Report Manager JCL Examples	2-50
Report Manager Single-Step JCL for Collating and/or Routing.....	2-50
Report Manager Three-Step JCL for Collating with or without Routing	2-52
Report Manager Three-Step JCL for Routing Only	2-56
Alternate Report Output Method JCL Examples	2-59
Alternate Report Output Methods Single-Step JCL.....	2-59
Alternate Report Output Methods Three-Step JCL.....	2-63

Chapter 3: VISION:Builder – IMS Database Interface and Retrieval

MARKDLI, DBDs, and PSBs.....	3-1
Sample File Definition	3-4
IMS Batch Region Execution of VISION:Builder.....	3-7
IMS BMP Region Execution of VISION:Builder	3-9
VISION:Builder Extended DL/I Support.....	3-10
Secondary Indexing	3-10
Non-Unique IMS Key/Search Field Names.....	3-10
Generic and Duplicate Root Keys.....	3-10

Chapter 4: VSAM User Files

ESDS and KSDS Files.....	4-1
ESDS and KSDS Alternate Index Paths.....	4-1
Alternate Index as a User File	4-3
Generic and Duplicate Keys	4-4

Chapter 5: Own-Code Facilities

Integrating M4OWN.....	5-2
Obtaining Space for Own-Code Routines	5-3
Interrupts and Linkage Considerations	5-4
Own-Code Hook Naming Conventions	5-5
VISION:Builder Own-Code Hook Flow	5-7

Own-Code Hook Descriptions	5-9
Own-Code Hook 10	5-9
Own-Code Hook 11	5-10
Own-Code Hook 20	5-11
Own-Code Hook 21	5-12
Own-Code Hook 30	5-14
Own-Code Hook 50	5-15
Own-Code Hook 51	5-16
Own-Code Hook 60	5-16
Own-Code Hook 61	5-17
Own-Code Hook 62	5-17
Own-Code Hook 63	5-17
Own-Code Hook 70	5-18
Own-Code Hook 91	5-20
Own-Code Hook 92	5-21
Own-Code Hook 93	5-22
Variable Length Fields with Own-Code	5-23
User I/O	5-23
Primary Parameter List	5-24
Communication Table.....	5-26
File Table.....	5-30
Key Table	5-33
Relationships Among Tables	5-36
Update-in-Place Example ④	5-37
COBOL Example of User I/O	5-37

Chapter 6: Checkpoint/Restart

Checkpoint Options	6-1
Taking Checkpoints in VISION:Builder.....	6-3
Checkpoint Files	6-3
Checkpointing at Time Intervals.....	6-3
Checkpointing on Record Count	6-4
Checkpointing Under Operator Control.....	6-4
Checkpointing at End of Volume.....	6-4
Multiple Checkpoint Options.....	6-4
Writing Checkpoints	6-5
OS/390 Samples of Checkpoint/Restart	6-5
IMS/ESA Checkpoint/Restart	6-8
IMS/ESA Requirements for Checkpointing	6-9
VISION:Builder Considerations for IMS/ESA Checkpoints	6-13
VISION:Builder Restrictions for IMS/ESA Checkpoint/Restart	6-14
IMS/ESA Checkpoints with DB2	6-14

Chapter 7: Batch Query Language Execution

Query Language Input and Output Files	7-1
Steps in the Execution of the Batch Query Language.....	7-2
Query Step.....	7-3
Glossary Step	7-5
Processing Step	7-5
Definition Step	7-6
Sort Step	7-6
Report Step	7-6
Sample Execution of Batch Query Language.....	7-6

Chapter 8: Batch Free-Form Input Execution

Job Control Language Requirements	8-1
---	-----

Chapter 9: VISION:Builder Online Execution

Using the VISION:Builder Executive Under TSO	9-1
Input and Output Devices	9-1
Furnishing Allocation Data	9-1
Using CLISTs.....	9-4
Messages Issued By OLX	9-4
OLX Commands.....	9-4
Data Set and Command Security Interfaces.....	9-5
Online Query Language.....	9-5
Online Free-Form Input	9-5
Input Mode and Edit Mode	9-5
Input of VISION:Builder Source Statements.....	9-6
OFI Commands	9-10
Sample Run	9-11

Chapter 10: Operating Characteristics

JCL Override Parameters	10-1
Concatenation of Input Data Sets	10-3
Linking to VISION:Builder	10-3
Calling Sequence	10-4
ddname	10-5
Exit Sequence	10-5
Sort Control Statements.....	10-5
Limiting the Number of Records Read During Input File Processing	10-7
Resource Optimization.....	10-7

Access Methods	10-9
Blocking Factor for ISAM Files.....	10-9
Non-VSAM Variable-Spanned Records	10-9

Chapter 11: Using VISION:Builder with DB2

TSO Attach Facility	11-1
IMS Attach Facility.....	11-2
CALL Attach Facility	11-3

Chapter 12: Common Library Access and Utilities

Using Multiple Common Libraries	12-2
Using Common Libraries on Shared DASD.....	12-2
Sharing Common Libraries on DASD.....	12-3
VISION:Builder Release 14.0 / COMLIB Release 4.5 Characteristics and Notes	12-4
Internal M4LIB Format.....	12-4
Common Library Utility Work File (M4WORK) Format	12-4
Common Library Data Set (M4LIB) Compression	12-5
The Common Library Service Program	12-6
MARKUTIL Initialization	12-6
MARKUTIL Dump and Restore	12-9
MARKUTIL Condense	12-11
MARKUTIL Copy and Merge	12-13
MARKUTIL and Multiple UC Statements	12-15
MARKINIT Common Library Initialization	12-16
MARKDUMP and MARKREST Common Library Dump and Restore.....	12-17
MARKCON Common Library Condense	12-20
Using Cataloged Procedures and Requests.....	12-21
Cataloged Procedure and Request Maintenance.....	12-21
Cataloged Procedure and Request Execution.....	12-21
Listing and Retrieving Common Library Items	12-22
Common Library Listing Operations.....	12-22
Common Library Retrieval Operations	12-22
Sample Source Statement Retrieval Runs.....	12-23
Fixed Syntax Statement Usage Considerations	12-24

Chapter 13: VISION:Builder HTML Document Style Customization

Index

Introduction

This book provides detailed information on the use of Advantage VISION:Builder[®] (hereafter referred to as VISION:Builder) and the COMLIB subsystem. This book covers operations, instructions for input to the system, and OS/390[®] run setups for all VISION:Builder models.

Each VISION:Builder system can be combined with options in a variety of ways to provide any user with the VISION:Builder processing functions needed to solve problems. Depending on configuration, these systems are upward compatible.

Since not all functions are available in every installation, you might find that some specifications described in this book are not operable with your system. When that is the case, VISION:Builder outputs an appropriate message.

Configuration Requirements

The VISION:Builder software system has several models grouped in two different series.

- The 4000 model series represents the VISION:Builder software system that allows users to develop fully functional applications. These applications include the transaction processing and master file updating facilities along with the entire set of information retrieval, selection, and manipulation functions used in conjunction with the multitude of reporting and data extraction capabilities.
- The 2000 model series represents the VISION:Builder software system subset known as VISION:Two[™]. This subset of VISION:Builder includes all the same information retrieval, selection, and manipulation functions that are used in conjunction with the multitude of reporting and data extraction capabilities that are part of the 4000 model series. However, the transaction processing and master file updating facilities are not included in the 2000 model series.

Organization and Standards for This Book

The term VISION:Builder is used to represent both the 4000 model series and the VISION:Two 2000 model series. The model number appears on the banner page that is output at the start of all source listing displays. Throughout this book, the symbol ⁴ is used to designate that the feature or function being described only applies to the VISION Builder 4000 model series.

The job control language (JCL) examples in this book follow a standard format. Uppercase characters are used for specifications that are thought to be usable as is. Lowercase characters are used for specifications, such as data set names and volume serial numbers, that are installation dependent in nature.

The following is a brief description of each chapter contained in this book.

- [Chapter 1, Introduction](#) is an introduction and overview of VISION:Builder application development. It also describes the minimum configuration required by VISION:Builder.
- [Chapter 2, VISION:Builder Runs, Run Control, and Execution JCL](#) describes the VISION:Builder run control group, the various VISION:Builder runs, and the OS/390 JCL to execute these runs.
- [Chapter 3, VISION:Builder – IMS Database Interface and Retrieval](#) describes using VISION:Builder and IMS™ Data Base Interface and Retrieval.
- [Chapter 4, VSAM User Files](#) describes the use of VSAM user files and VISION:Builder.
- [Chapter 5, Own-Code Facilities](#) describes how to use Own-Code and User I/O facilities.
- [Chapter 6, Checkpoint/Restart](#) describes Checkpoint/Restart with VISION:Builder.
- [Chapter 7, Batch Query Language Execution](#) describes the execution of VISION:Builder in a Batch Query Language environment.
- [Chapter 8, Batch Free-Form Input Execution](#) describes how to use VISION:Builder Batch Free-Form Input.
- [Chapter 9, VISION:Builder Online Execution](#) describes how to use VISION:Builder Online Executive under TSO.
- [Chapter 10, Operating Characteristics](#) describes some operating characteristics of VISION:Builder.
- [Chapter 11, Using VISION:Builder with DB2](#) describes using VISION:Builder with DB2®.
- [Chapter 12, Common Library Access and Utilities](#) describes the maintenance of the common library.

Overview of VISION:Builder Application Development

VISION:Builder is a general purpose software system for the design, implementation, and operation of data processing applications.

VISION:Builder is able to input and output a variety of files. For a detailed discussion of input and output files, refer to the [VISION:Builder Reference Guide](#).

[Figure 1-1](#) illustrates the input files processed by VISION:Builder. [Figure 1-2](#) illustrates the output files produced by VISION:Builder during the processing step while [Figure 1-3](#) illustrates the output files produced during the report step.

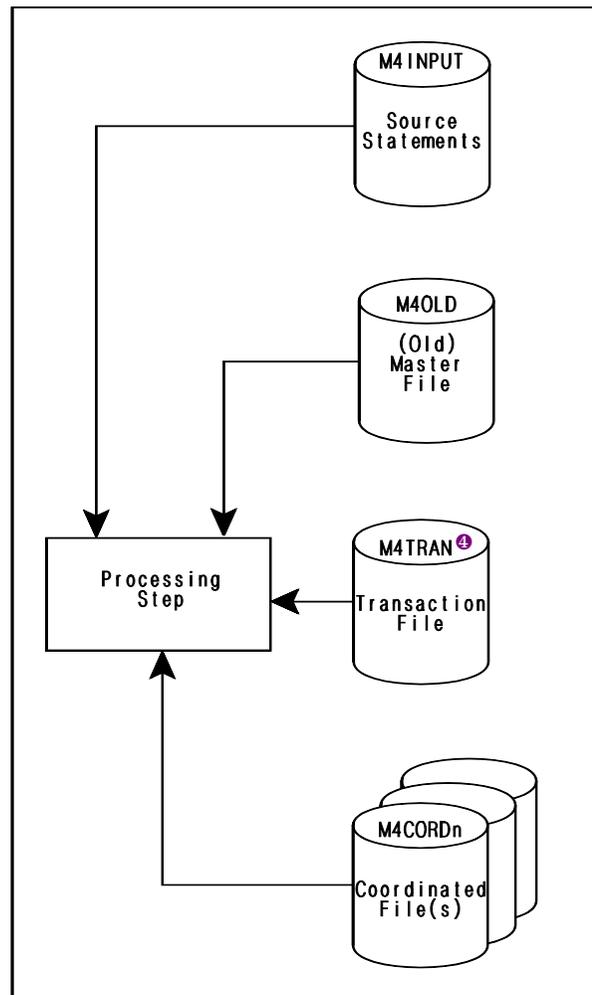


Figure 1-1 Input Files Processed by VISION:Builder

Note: For M4CORDn in [Figure 1-1](#), n can be a value from 0 to 9.

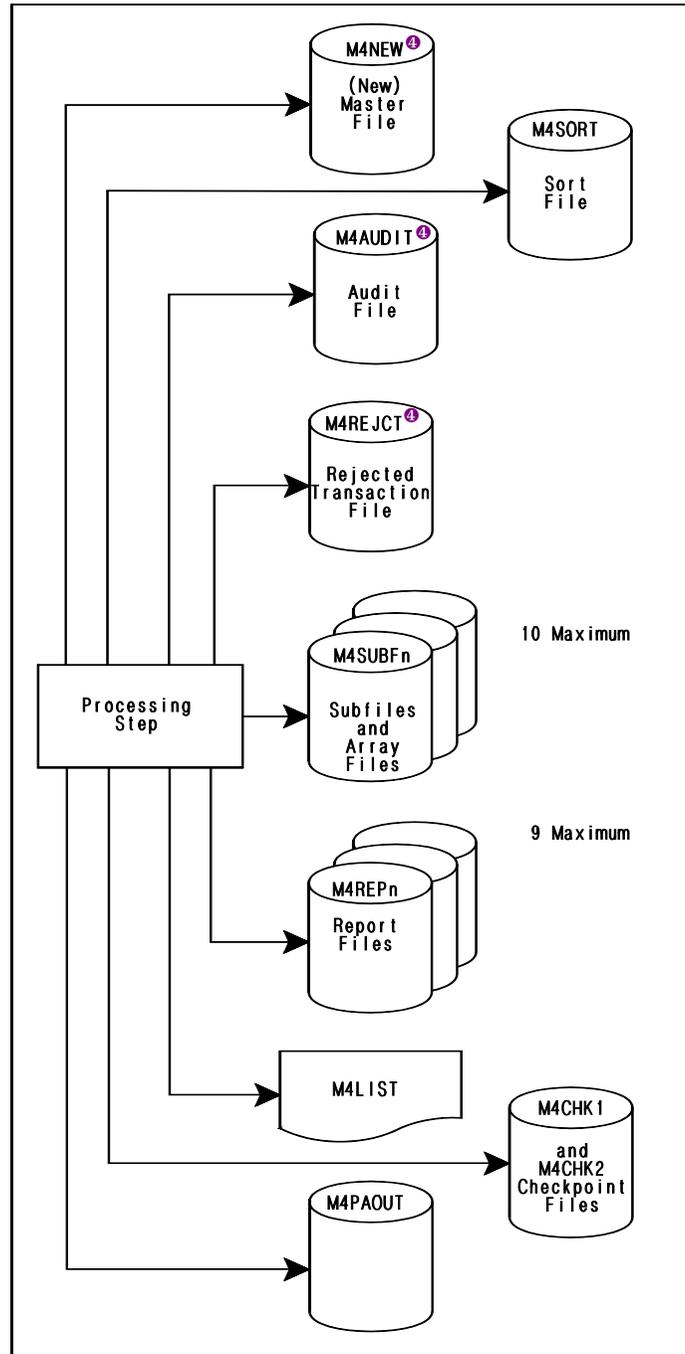


Figure 1-2 Output Files Produced During Processing

Note: For M4SUBFn and M4REPn in [Figure 1-2](#), n can be a value from 0 to 9.

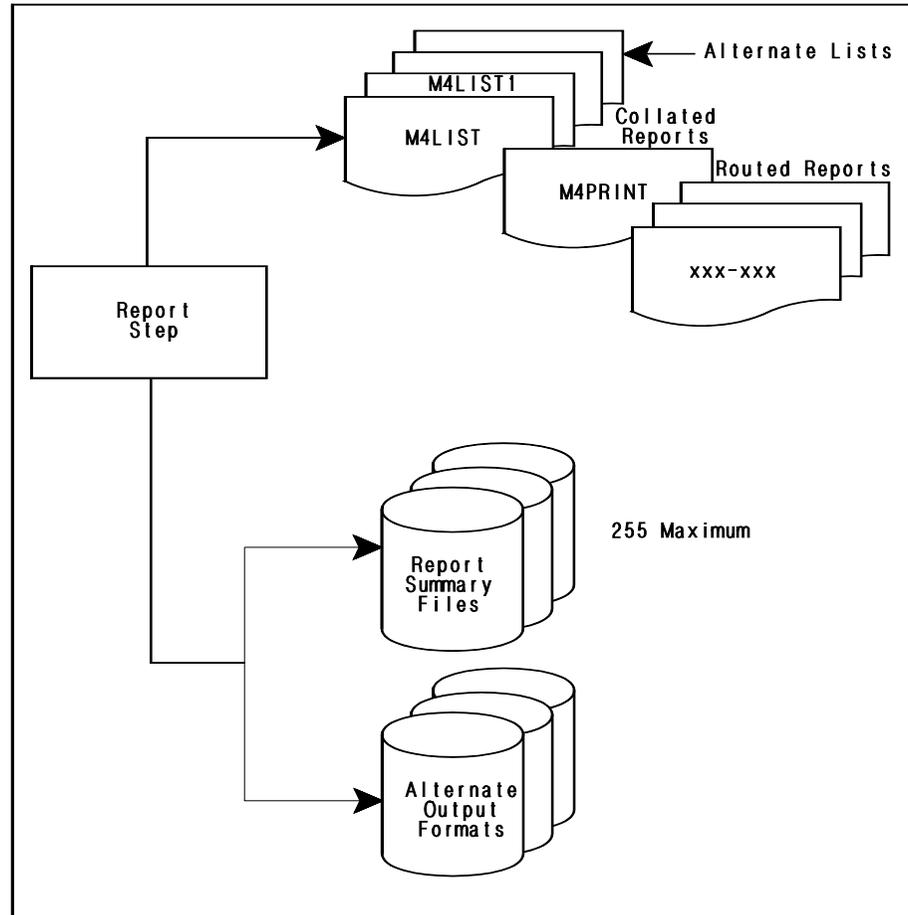


Figure 1-3 Output Files Produced During Report Step

Definitions

Any file or data structure, including arrays, tables, and transaction definitions, input to a VISION:Builder application run must first be defined to VISION:Builder. You define data structures to VISION:Builder by inputting definition source statements to a definition/maintenance run.

Note: You can also define master and coordinated files instream in a processing run (refer to the [VISION:Builder Reference Guide](#)).

The definition/maintenance run checks the source statements for errors and catalogs error free definitions in the common library (M4LIB), which is located on a direct access device (DASD). The printed output of a definition run is a glossary, a formatted listing of the data structure defined to VISION:Builder. [Figure 1-4 on page 1-6](#) illustrates the definition/maintenance run.

Before defining any data to VISION:Builder, you must initialize the common library. See [Chapter 12, Common Library Access and Utilities](#) for more information on how to initialize the common library.

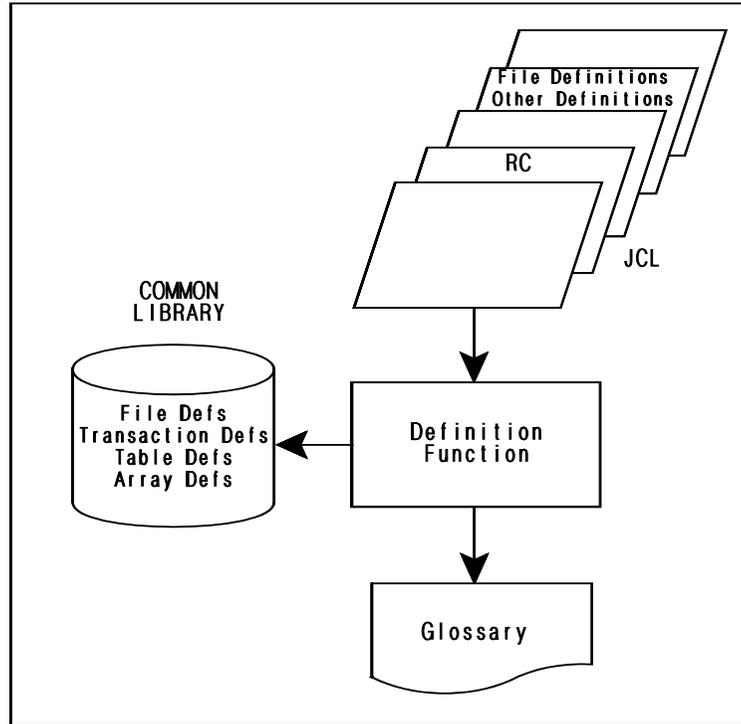


Figure 1-4 A Definition/Maintenance Run

Processing Data

The processing of data by VISION:Builder occurs in an application run. An application run that generates report output typically consists of three functions: processing, sorting, and reporting.

You can combine the three functions into a single step where the report data is sorted internally rather than being passed to a separate sort step and then a report step. This method of processing an application run is known as single-step processing, which optimizes processing and saves external I/O operations.

Application runs can be processed by a three-step VISION:Builder processing method. The processing step is first and it includes the decoding of the VISION:Builder source statements, reading of input files, outputting new files, and, often, producing a report file (or files). The final two steps, sorting and reporting, are for the processing of the report file and possibly producing report summary files.

[Figure 1-5 on page 1-7](#) illustrates the conceptual flow of an application run. [Figure 1-6 on page 1-8](#) illustrates single-step and three-step application runs.

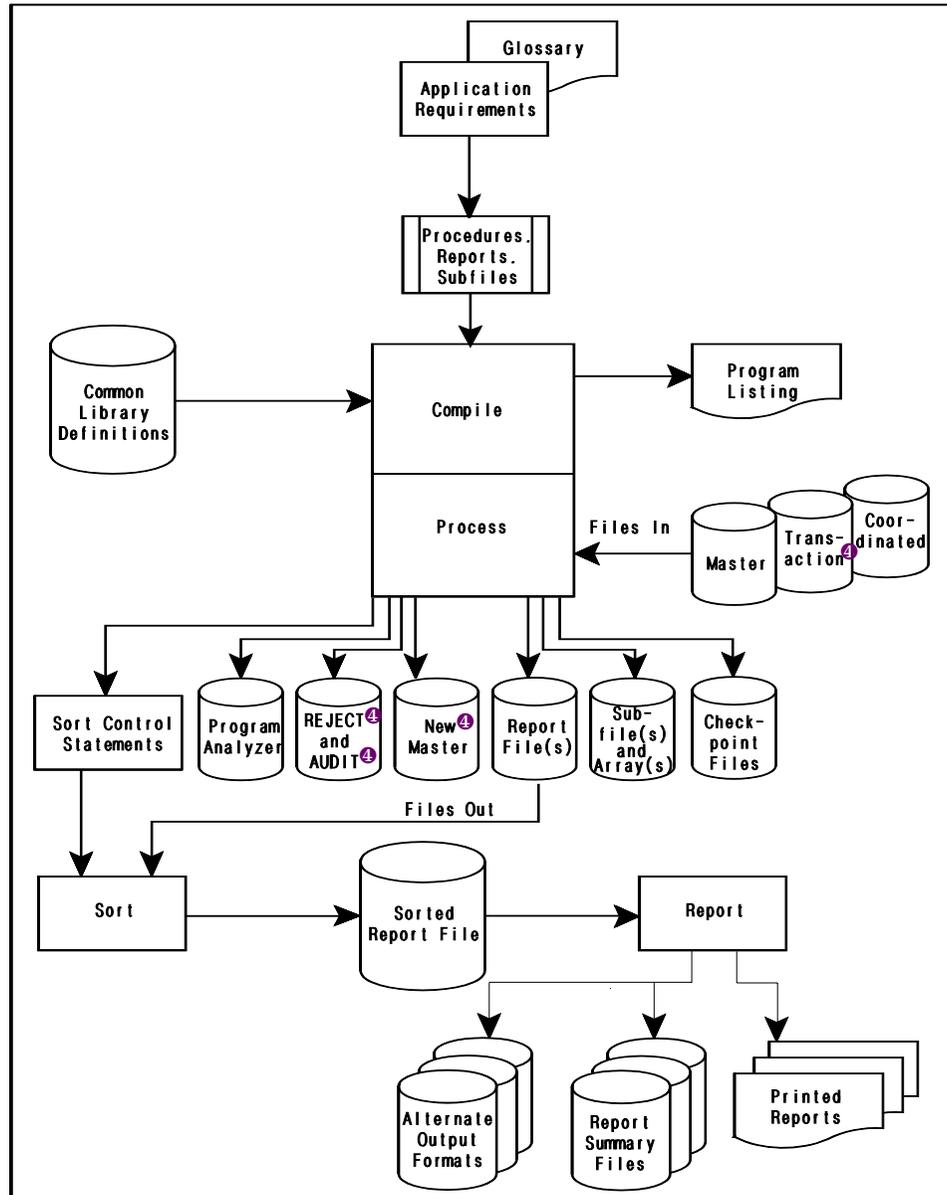


Figure 1-5 The Application Run

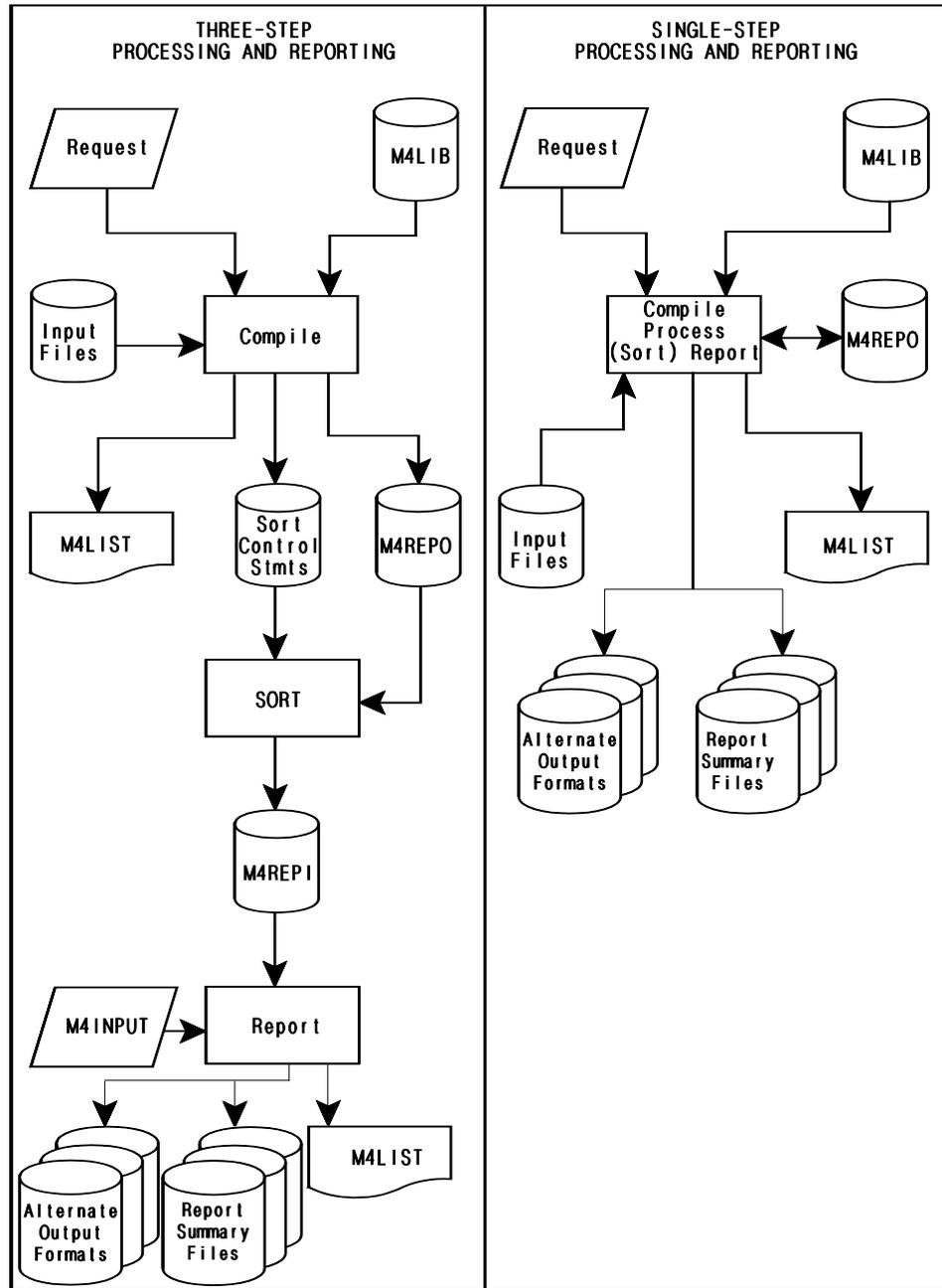


Figure 1-6 Three-Step vs. Single-Step Processing and Reporting

Note: In [Figure 1-6](#), input files include any VISION:Builder file that can be considered an input file, such as M4OLD, M4TRAN, ⁴M4CORDn, and M4INPUT.

VISION:Builder Runs, Run Control, and Execution JCL

This chapter describes VISION:Builder runs, run control, execution JCL, and their relationships to each other. It is divided into the following sections:

- [Relationship Between Run Control, VISION:Builder Processing, and JCL on page 2-1](#)
- [Files and Their DDnames in a VISION:Builder Run on page 2-2](#)
- [Run Control Group on page 2-5](#)
- [The Definition Maintenance Run on page 2-8](#)
- [The Application Run on page 2-9](#)
- [Run Control and Execution JCL for Sample VISION:Builder Applications on page 2-17](#)
- [Update-in-Place \(Single-Step No-Sort\)¼ on page 2-48](#)
- [Report Manager JCL Examples on page 2-50](#)
- [Alternate Report Output Method JCL Examples on page 2-59](#)

Relationship Between Run Control, VISION:Builder Processing, and JCL

The first statement or set of statements in any VISION:Builder run is the run control group. These statements perform the following functions:

- They define the master, transaction,⁴ and report files for the run, and allow you to control run-dependent parameters. The CONTROL statement allows you to select a variety of options that affect how VISION:Builder executes.
- The run control group acts as a bridge between the files (described by JCL) and the VISION:Builder processing performed. Based on the files you specify in the run control group, VISION:Builder locates the appropriate JCL statements that define the files to the operating system.

Files and Their DDnames in a VISION:Builder Run

This book illustrates typical JCL that you need to execute VISION:Builder under IBM's OS/390 operating system. VISION:Builder accepts most sequential or indexed-sequential files. The files you use can have standard labels or they can be unlabeled. They can exist on any standard I/O device.

Provisions are made on the VISION:Builder run control statements for indication of the label type for all of the files that can be used in your system. The JCL you provide to assign files must agree with the run control specifications.

Job Control Statements

The following table contains a brief description of each type of OS/390 JCL statement. The table on page [2-3](#) shows the ddnames for all of the VISION:Builder files. See [Chapter 10, Operating Characteristics](#) for additional JCL information.

Statement		Format	Description	
JOB	//jobname	JOB	(accounting information), name, msglevel=1	The job name is one to eight alphanumeric characters.
JOBLIB	//JOBLIB	DD	(parameters describing the data sets where the VISION:Builder and COMLIB programs reside)	
EXEC	//stepname	EXEC	PGM=MARKIV	
DD	//ddname	DD	(parameters describing the data set)	The choice of DSNAME is entirely arbitrary. The ddname must be one of the entries specified in the following table, except where otherwise noted.

Description	DDName	VISION:Builder DDName Assignments/Comments
Source Input (Fixed Syntax or ASL)	M4INPUT	Assigned to a standard I/O device which is: <ol style="list-style-type: none"> 1. physical sequential or a member of a partitioned data set. 2. blocked one or more 80-character records per block.
Object Input	M4OWN	Must be a load module in a partitioned data set.
Source Listing and Reports	M4LIST	Usually assigned to a printer, but can be any data set that is: <ol style="list-style-type: none"> 1. physical sequential. 2. blocked one or more 133-character records per block (or the M4PARAMS value).
Alternate List File	M4LIST1 (or any unique name)	Any data set that is: <ol style="list-style-type: none"> 1. physical sequential. 2. blocked one or more 133-character records per block (or the M4PARAMS value).
Old Master In	M4OLD	A physical sequential or VSAM data set, IMS database, or DB2 tables.
New Master Out	M4NEW	A physical sequential or VSAM data set, or IMS database. ⁴
Transaction In	M4TRAN	A physical sequential or VSAM dataset, or IMS database. ⁴
Audit File Out	M4AUDIT	A physical sequential dataset. ⁴
Report File Out	M4REPO	A physical sequential dataset.
Report File In	M4REPI	A physical sequential dataset.
Coordinated File n	M4CORDn	A physical sequential or VSAM data set (n=1-9), IMS database, or DB2 tables.
Subfile n	M4SUBFn	A physical sequential or VSAM dataset (n=0-9), IMS database, or DB2 tables.

Description	DDName	VISION:Builder DDName Assignments/Comments
Sort Control	M4SORT	Any data set which is: <ol style="list-style-type: none"> 1. physical sequential. 2. blocked one or more 80-character records per block.
Common Library	M4LIB	A direct-access or VSAM data set. DISP=SHR may be specified on the DD statement for M4LIB; if this is done, VISION:Builder will ensure the integrity of the library. Multiple common libraries are supported for processing runs only. In this case, the ddname is M4LIBn where n is a number 1-9.
Rejected Transaction File	M4REJCT	Must be a physical sequential data set. ⁴
Alternate Report File Out	M4REPn	A physical sequential data set (n = 2-9).
Work File	M4WORK	A physical sequential data set.
Source Statement Out	M4SSOUT	Assigned to a standard I/O device that is: <ol style="list-style-type: none"> 1. physical sequential. 2. blocked one or more 80-character records per block.
Program Analyzer	M4PAOUT	A physical sequential data set.
Checkpoint File	M4CHKn	A file containing checkpoint information (n = 1-2).
Report Summary File	Any unique name	A physical sequential data set, blocked with variable length records.
Report Manager Reports	M4PRINT (or any unique name)	Usually assigned to a printer, but can be any data set that is: <ol style="list-style-type: none"> 1. physical sequential. 2. blocked one or more 133-character records per block (or the M4PARAMS value).

Description	DDName	VISION:Builder DDName Assignments/Comments
Extended Subfile Output	Any unique name	A physical sequential or VSAM data set, IMS database, or DB2 tables.
Alternate Report Output Method	Any unique name	A physical sequential data set. Attributes dependent upon method selected.
Captured Fixed-syntax from ASL	M4FIXED	Assigned to a standard I/O device that is: <ol style="list-style-type: none"> 1. physical sequential. 2. blocked one or more 80-character records per block.

Run Control Group

The run control group is the first set of statements in the source input data set in every VISION:Builder run. These statements may be coded either in ASL or in the traditional fixed-syntax. The ASL run control group consists of the following statements (refer to the [VISION:Builder ASL Reference Guide](#) for a complete description of these statements):

CONTROL	Application parameters and controls - must be first statement.
FILE MASTER	Master file parameters and controls.
FILE CORDn	Coordinated file and external array parameters and controls.
FILE TRAN	Transaction file parameters and controls.
FILE REJECT	Rejected transaction file parameters.
FILE AUDIT	Audit file (deleted master file record) parameters.
FILE SUBFn	Subfile parameters.
FILE REPORT	Primary report file parameters.
FILE REPn	Additional report file parameters.
ARRAY	Internal array parameters.
WORK	Working storage parameters.
LINKAGE	Linkage section parameters.

ROUTE	Report routing parameters and controls.
COLLATE	Report collating parameters and controls.
CHECKPOINT	Checkpoint parameters and controls. See Chapter 6, Checkpoint/Restart .
CATALOG	Request cataloging controls.
TRACK	Catalog item tracking parameters.
OVERRIDE	DD Name override parameters.
MULTILIB	Multiple M4LIB ordering parameters.
OWNCODE	Own Code parameters. See Chapter 5, Own-Code Facilities .
LISTCNTL	Program output listing controls.
DOCUMENT	Application documentation (Program Analyzer) controls.
LISTLIB GLOSSARY	Catalog content glossary listing controls.
LISTLIB NAMES	Catalog content names listing controls.
RETRIEVE	Catalog content retrieval controls.
DEBUG	Debugging output controls.
COPY	Copy commands from an alternate input stream.

The fixed-syntax run control group consists of the following statements (refer to the [VISION:Builder Reference Summary](#) for a complete description of these statements):

- RC The run control (RC) statement specifies the type of run (processing, definition, or report). For a processing run, the CONTROL statement specifies the master, transaction, ⁴ and other input and output files for the run, and allows you control of run-dependent parameters. The CONTROL statement must be the first statement in the M4INPUT data set (except in a MARKUTIL run). The balance of the run control statements are optional; if used, they must directly follow the CONTROL statement.
- DB The DB statement specifies the name of the run data group. There must be one DB statement for each run data group you define. Run data groups are VISION:Builder run control statements that are saved in the common library to be used during VISION:Builder processing.

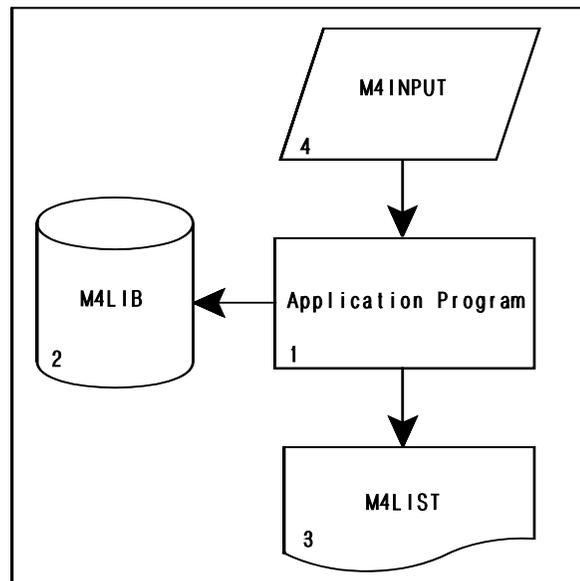
-
- RF The RF statement specifies files used other than the master file specified in a CONTROL statement. Use the RF statement to specify coordinated files, output data or subfiles, arrays, and alternate report files.
- LB The LB statement is used to assign alias names to segments, fields, or ddnames for files within a run data group.
- RA The RA statement allows you to overdefine an array with a secondary definition, allowing the same array to be referenced multiple ways.
- RP The run parameters (RP) statement supplies parameters that are effective only during the run where they are supplied. They provide a means of specifying execution time parameters.
- RG⁴ The transaction groups (RG) statement specifies the transaction groups to be used in a particular run.
- RT The relational tables (RT) statement provides the DB2® or SQL/DS™ table names and controls table updating for a particular run.
- WH The WH statement provides the option of extending the selection criteria of logical relationship (LR) statements for segments in a relational table. (This applies only if your system has the relational support option installed.)
- OC The own-code (OC) statement is used when user supplied coding is to receive control at selected points during the run. Own-code is discussed in [Chapter 5, Own-Code Facilities](#).
- CP The checkpoint (CP) statement is included in runs that need checkpoint/restart capabilities. Checkpoint/restart is discussed in [Chapter 6, Checkpoint/Restart](#).
- PA The program analyzer (PA) statement is included in runs to obtain program documentation and aid in debugging your VISION:Builder program.
- IT The item tracking (IT) statement is used to track items maintained in the common library. The three types of information maintained are:
- The date and time an item on M4LIB is created, updated, or retrieved for use. VISION:Builder maintains this information automatically.
 - A field that identifies the entity that caused an item to be created and another field that identifies the entity that caused an item to be updated. You supply this information.
 - A retention period/expiration date field that specifies either the length of time a temporarily cataloged item is to be retained in the library or the date the item expires. You supply this information.

The Definition Maintenance Run

You must define an input file before it can be processed by VISION:Builder. File, transaction, ⁴ table, and array definition statements are input to a VISION:Builder definition/maintenance run. Multiple file, transaction, table, and array definitions can be input to the same definition/maintenance run. Once cataloged, the definitions are available to any processing run when the defined input file is

specified on an RC, RG, ⁴ or RF statement for that run. You only need to reference a table for it to be available for processing. [Figure 2-1 on page 2-8](#) describes the flow and JCL for a definition/maintenance run.

The common library must be initialized before any VISION:Builder runs can be made. It is normally initialized only once. See [Chapter 12, Common Library Access and Utilities](#) for more information on how to initialize the common library.



Notes

```

//          JOB (accounting information)
/** JCL FOR A DEFINITION RUN **
//JOBLIB DD DSN=your.builder.loadlib,DISP=SHR
1 //def EXEC PGM=MARKIV,REGION=1536K
2 //M4LIB DD DSN=your.m4lib,DISP=SHR
3 //M4LIST DD SYSOUT=a
4 //M4INPUT DD *
CONTROL
** VISION:Builder DEFINITION SOURCE STATEMENTS **
/*
//
  
```

Figure 2-1 Flow and JCL for a Definition/Maintenance Run

An explanation of the numbered statements in [Figure 2-1](#) follows:

Notes	Explanation
1	The library maintenance functions of VISION:Builder are executed in a file definition run. These functions process source statements from the M4INPUT data set, checking for syntactical errors.
2	Syntactically correct definitions are cataloged or updated in the common library for later use in the processing of applications.
3	This DD statement defines the location of the system output device for this job. The M4LIST output from a file definition run includes a listing of the source statements in the M4INPUT data set, diagnostic messages, and glossaries of any definitions cataloged or updated on the common library (or requested for listing) in this definition run.
4	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements. The CONTROL statement is followed by TRACK, COPY, or ;;ENDASL statements. You can include valid fixed-syntax file definition statements, grouped by file name, after the ;;ENDASL statement.

The Application Run

A VISION:Builder application run that generates report output usually consists of three functions: processing, sorting, and reporting. Requests allow you to specify input, validation, manipulation, or selection of data to be used or output. Using these specifications, VISION:Builder processes the input files and produces reports, report summary files, or subfiles.

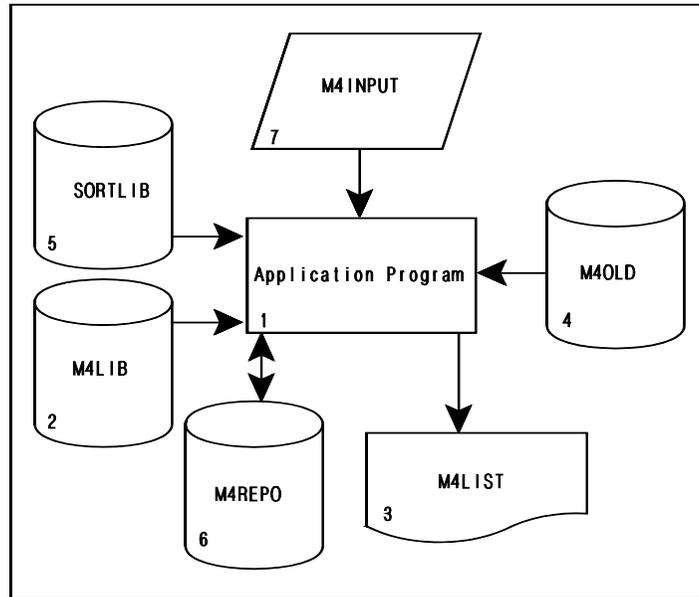
The next two subsections describe how the three function process can be executed in three separate steps or as a single step.

Single-Step Process-Sort-Report Run

VISION:Builder allows you to optimize I/O processing and CPU usage when generating reports through the use of single-step processing, where the report file is sorted internally rather than being passed to a sort step and to a report step. This is a more efficient process and reduces the amount of I/O for sorting and reporting.

The CONTROL statement associated with this process is the same as the three-step process listed in [Three-Step Process-Sort-Report Run on page 2-11](#), except for the absence of SORT EXTERNAL on the CONTROL statement. The JCL is similar to a processing run, with the following changes: M4SORT JCL can be eliminated (unless alternate report files are used), sort work JCL must be added, and DD

statements for SORTLIB and SYSOUT are required. [Figure 2-2 on page 2-10](#) shows the flow and JCL for a single-step processing run. Notice that a run parameter (RP) statement has been coded.



Notes

```

//          JOB (accounting information)
//* JCL FOR SINGLE-STEP (PROCESS-SORT-REPORT) **
//JOBLIB   DD DSN=your.builder.loadlib,DISP=SHR
//step    EXEC PGM=MARKIV,REGION=1536K
1         //M4LIB   DD DSN=your.m4lib,DISP=SHR
2         //M4LIST  DD SYSOUT=a
3         //M4OLD   DD DSN=old.master.file,DISP=(OLD,KEEP)
4         //SYSOUT  DD SYSOUT=a
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
5         //SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
6         //M4REPO  DD DSN=your.m4repo,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,(n,n))
7         //M4INPUT DD *
CONTROL SORTSIZE 200K
FILE MASTER INPUT, NAME . . .
FILE REPORT
.
.
.
/*
//
  
```

Figure 2-2 Flow and JCL for Single-Step Application Run (Process-Sort-Report)

An explanation of the numbered statements for [Figure 2-2](#) follows:

Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors, and generates the necessary sort control statements. The execution of VISION:Builder is controlled by run control statements that specify the files and DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name required for the file definition.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements, diagnostic messages, and the report(s).
4	M4OLD defines the master file and is the primary data input to the processing step in this run.
5	SORTLIB defines the location of the system sort program.
6	M4REPO is the file containing the report command language necessary to build the report from the processing step. The report data records are passed to the sort process.
7	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements. The CONTROL statement is followed by other run control statements and any valid requests. You may need to code SORTSIZE on the CONTROL statement if SORT INTERNAL (the default if there is no SORT keyword) has been specified and SORTSIZE in M4PARAMS is not adequate. The SORTSIZE parameter indicates the amount of main storage VISION:Builder allocates for the sort.

Three-Step Process-Sort-Report Run

VISION:Builder application runs that generate reports can be invoked using a three-step process. Each step can also be executed as a separate job. The first is the processing step, which includes decoding of the VISION:Builder source

statements, reading input files, outputting new files (including master file), ⁴ and production of VISION:Builder report files (M4REPO, M4REP2-9). Sort control statements are produced automatically by VISION:Builder during this step. The sort control statements and M4REPO are input to the sort step, as SYSIN and SORTIN, respectively.

The sort step executes a standard IBM OS compatible sort program to sort the report file according to the VISION:Builder sort parameters. The output is a sorted report file, which is the input for the third (report) step as M4REPI.

The report step generates the report using the sorted report file (M4REPI) to perform a variety of functions, including:

- calculate summaries, percents, and ratios
- group information using control break specifications
- generate title information for each page

Figure 2-3 and Figure 2-4 list the flow and JCL for a typical application run.

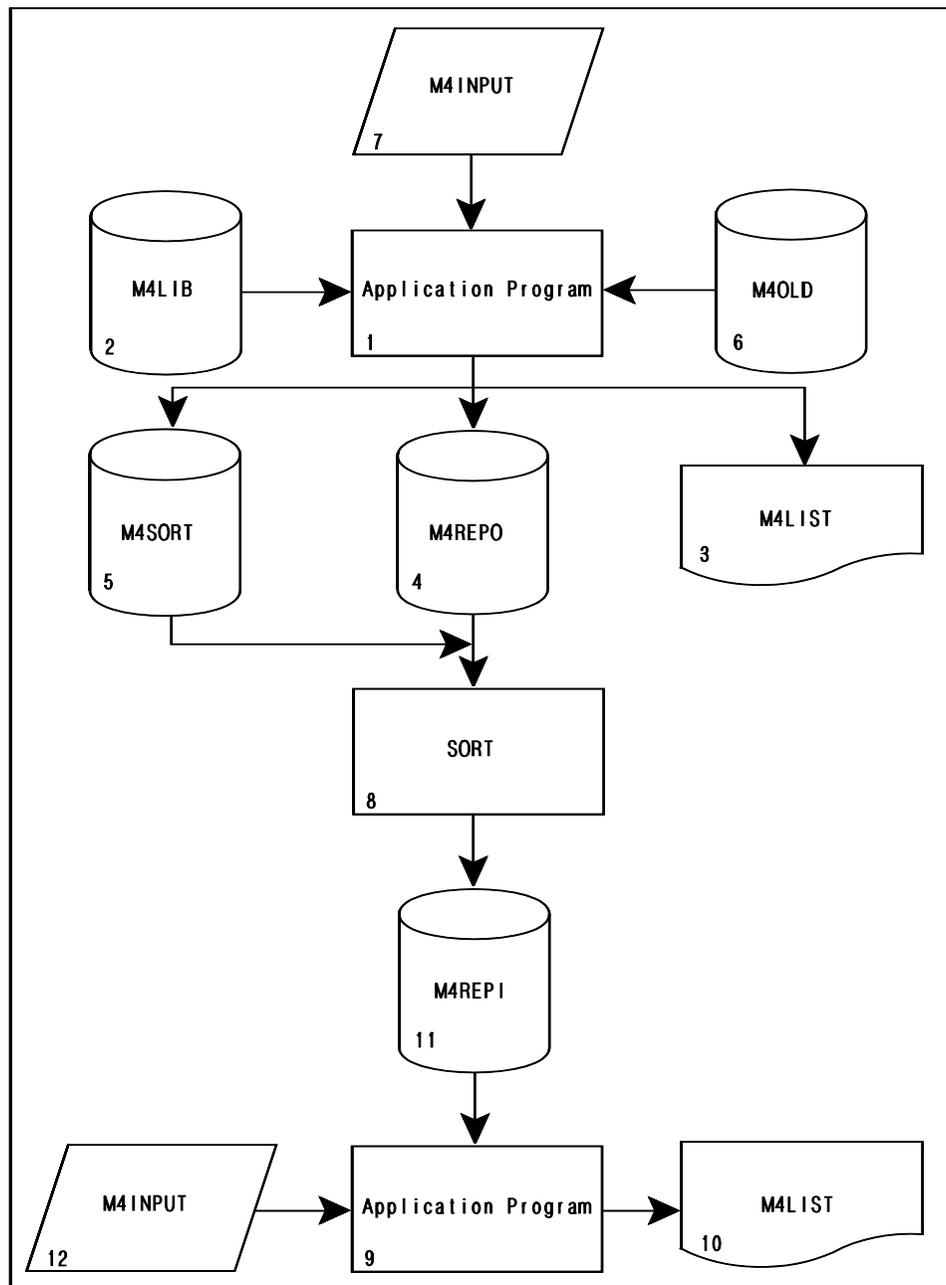


Figure 2-3 Flow of a Three-Step Application Run (Process-Sort-Report)

Notes

```

//          JOB(accounting information)
/* JCL FOR A THREE-STEP APPLICATION RUN **
//JOBLIB   DD DSN=your.builder.loadlib,DISP=SHR
1 //stepa   EXEC PGM=MARKIV,REGION=1536K
2 //M4LIB   DD DSN=your.m4lib,DISP=SHR
3 //M4LIST  DD SYSOUT=*
4 //M4REPO  DD DSN=your.m4repo,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,(n,n))
5 //M4SORT  DD DSN=sort.file,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,n)
6 //M4OLD   DD DSN=old.master.file,DISP=(OLD,KEEP)
7 //M4INPUT DD *
CONTROL SORT EXTERNAL
FILE MASTER INPUT, NAME . . .
FILE REPORT
.
.
/*
8 //sort    EXEC PGM=SORT
//SYSIN   DD DSN=sort.file,DISP=(,PASS)
//SORTIN  DD DSN=your.m4repo,DISP=(OLD,DELETE)
//SORTOUT DD DSN=your.m4repi,DISP=(NEW,PASS),
//          UNIT=sysda,SPACE=(TRK,(n,n))
//SYSOUT  DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
/*
9 //report  EXEC PGM=MARKIV,REGION=1536K
10 //M4LIST  DD SYSOUT=*
11 //M4REPI  DD DSN=your.m4repi,DISP=(OLD,DELETE)
12 //M4INPUT DD *
CONTROL
FILE REPORT
/*
//

```

Figure 2-4 JCL for a Three-Step Application Run (Process-Sort-Report)

An explanation of the numbered statements for [Figure 2-3](#) and [Figure 2-4](#) follows:

Note: SORTOUT should not have DCB information on the JCL.

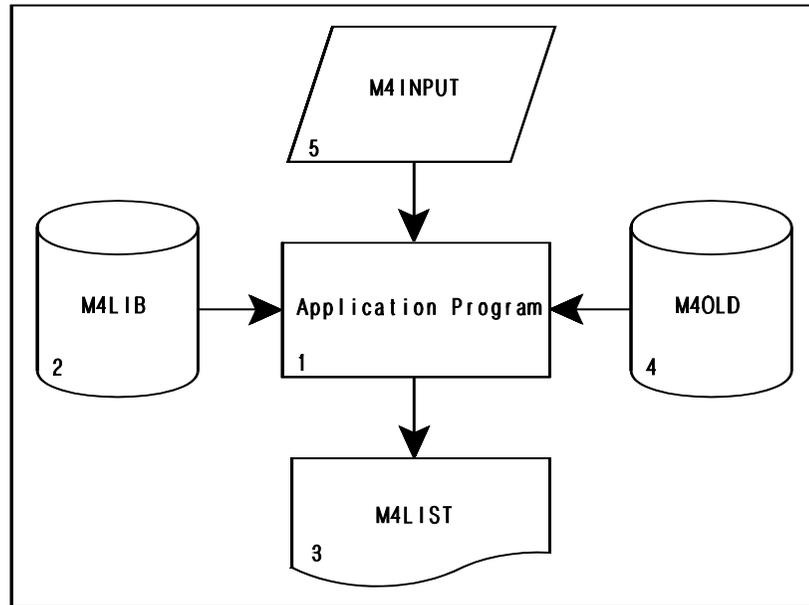
Note	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors, and generates the necessary sort control statements. The execution of VISION:Builder is controlled by run control statements that specify the files and DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements, diagnostic messages and the report(s).

Note	Explanation
4	M4REPO is the file containing the VISION:Builder report file output from the processing step. The data is used in subsequent sort and report generation steps.
5	The M4SORT file contains the sort control statements for the sort program.
6	M4OLD defines the old master file and is the primary data input file to the processing step in this run.
7	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements and you must code SORT EXTERNAL to specify a three-step run.
8	This statement calls in and executes the system sort program.
9	VISION:Builder is invoked to generate the report.
10	This M4LIST is output from the report step and includes a listing of the CONTROL statements, report messages and the reports.
11	Defines the location of the sorted report file (M4REPI), which is output by the sort step and the input to the report step.
12	This M4INPUT indicates to VISION:Builder that only the report step has to be executed.

Single-Step No-Sort Report Run

In a single-step processing run that does not require sorting, M4REPO and sorting statements can be eliminated. Sorting is not required when the fields will be printed on the report in the same order as the input file and only a single report is being produced. The CONTROL statement for this process is the same as the CONTROL statement listed in [Single-Step Process-Sort-Report Run on page 2-9](#),

except for the addition of the SORT NONE. The SORT NONE indicates to VISION:Builder that no sort is required. [Figure 2-5](#) shows the flow and JCL for a single-step processing no-sort run.



Notes

```

//          JOB(accounting information)
// * JCL FOR A SINGLE-STEP (NO SORT) RUN **
//JOBLIB   DD DSN=your.builder.loadlib,DISP=SHR
1 //step   EXEC PGM=MARKIV,REGION=1536K
2 //M4LIB  DD DSN=your.m4lib,DISP=SHR
3 //M4LIST DD SYSOUT=A
4 //M4OLD  DD DSN=old.master.file,DISP=(OLD,KEEP)
5 //M4INPUT DD *
CONTROL SORT NONE, REPTSIZE 200K
FILE MASTER INPUT, NAME . . .
FILE REPORT
.
.
/*
//

```

Figure 2-5 Flow and JCL for Single-Step Application Run (No-Sort)

An explanation of the numbered statements in [Figure 2-5](#) follows:

Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors, selects data for the report, and passes the report records directly to the report phase. The execution of VISION:Builder is controlled by run control statements that specify the files and DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements, diagnostic messages, and the report(s).
4	M4OLD defines the old master file and is the primary data input file to the processing step in this run.
5	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements. You must code SORT NONE on the CONTROL statement for a single-step no-sort run. The CONTROL statement is followed by other run control statements and any valid requests. Only one request can produce a report when SORT NONE is specified.

When resource optimization is used and it is a no-sort run, a CONTROL statement may need to be coded with REPTSIZE, if M4PARAMS is not adequate.

Run Control and Execution JCL for Sample VISION:Builder Applications

The following sections illustrate JCL examples that are used in conjunction with various types of VISION:Builder runs. It is also possible to use varying combinations of VISION:Builder files, depending on your application needs.

These JCL examples show different types of input and output files and the use of the run control group. You can find the descriptions for a simple application run, either single-step or three-step method, in [Single-Step Process-Sort-Report Run on page 2-9](#) or [Three-Step Process-Sort-Report Run on page 2-11](#).

Alternate Report Files

A VISION:Builder processing step can produce up to nine report files (M4REPO, M4REP2-9) to facilitate multiple report forms and multiple report destinations. A report file contains reporting, sorting, and printing information. Each report file requires a separate sort step and a separate report step.

[Figure 2-6](#) and [Figure 2-7](#) show the flow and JCL for a three-step (process-sort-report) alternate report files application run. An additional files (RF) statement must be coded for each additional report file (M4REP2-9). Alternate report files can be produced in a single-step run.

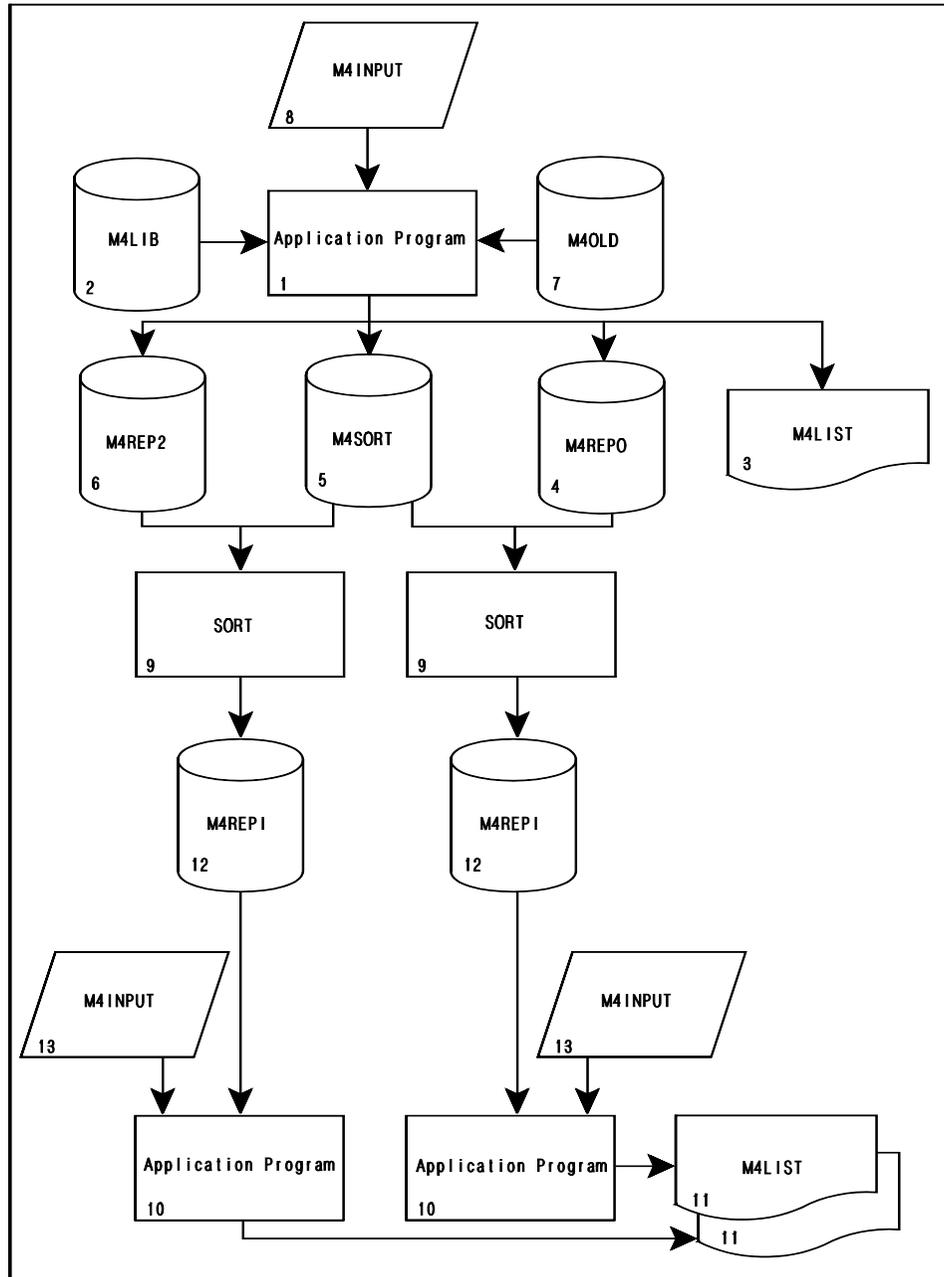


Figure 2-6 Flow for a Three-Step (Process-Sort-Report) Alternate Report Files Application Run

Notes

```

//          JOB(accounting information)
// * JCL FOR THREE-STEP ALTERNATE FILES APPLICATION RUN **
//JOBLIB   DD DSN=your.builder.loadlib,DISP=SHR
1 //step    EXEC PGM=MARKIV,REGION=1536K
2 //M4LIB   DD DSN=your.m4lib,DISP=SHR
3 //M4LIST  DD SYSOUT=a
4 //M4REPO  DD DSN=your.m4repo,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,(n,n))
5 //M4SORT  DD DSN=sort.file,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,n)
6 //M4REP2  DD DSN=your.m4rep2,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,n)
7 //M4OLD   DD DSN=old.master.file,DISP=(OLD,KEEP)
8 //M4INPUT DD *
CONTROL SORT EXTERNAL
FILE MASTER INPUT, NAME . . .
FILE REPORT
FILE REP2 NAME . . .
.
.
9 // *
//sorta   EXEC PGM=SORT
//SYSIN   DD DSN=sort.file,DISP=(OLD,PASS),
//SORTIN  DD DSN=your.m4repo,DISP=(OLD,DELETE)
//SORTOUT DD DSN=your.m4repi,DISP=(NEW,PASS),
//          UNIT=sysda,SPACE=(TRK,(n,n))
//SYSOUT  DD SYSOUT=a
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
10 // *
11 //report  EXEC PGM=MARKIV,REGION=1536K
12 //M4LIST  DD SYSOUT=a
13 //M4REPI  DD DSN=your.m4repi,DISP=(OLD,delete)
//M4INPUT DD *
CONTROL
FILE REPORT
9 // *
//sortb   EXEC PGM=SORT
//SYSIN   DD DSN=sort.file,DISP=(OLD,DELETE)
//SORTIN  DD DSN=your.m4rep2,DISP=(OLD,DELETE)
//SORTOUT DD DSN=your.m4repi,DISP=(NEW,PASS),
//          UNIT=sysda,SPACE=(TRK,(n,n))
//SYSOUT  DD SYSOUT=a
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
10 // *
11 //rept    EXEC PGM=MARKIV,REGION=1536K
12 //M4LIST  DD SYSOUT=a
13 //M4REPI  DD DSN=your.m4repi,DISP=(OLD,DELETE)
//M4INPUT DD *
CONTROL
FILE REPORT
//

```

Figure 2-7 JCL for a Three-Step (Process-Sort-Report) Alternate Report Files Application Run

The numbered statements in [Figure 2-7](#) represent those dealing with alternate files. An explanation of the numbered statements follows:

Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors, and generates the necessary sort control statements. The execution of VISION:Builder is controlled by run control statements that specify the files and DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements and diagnostic messages.
4	M4REPO is the file containing the VISION:Builder report file output from the processing step. The data is used in subsequent sorting and reporting steps.
5	The M4SORT file contains the sort control statements needed to sort the report file generated by VISION:Builder in the sort program.
6	M4REP2 is the file containing the alternate report file output from the processing step. The data is sorted and reported separately from M4REPO data.
7	M4OLD defines the old master file and is the primary data input file to the processing step in this run.
8	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements and additional FILE statements must be specified as required. The CONTROL statement is followed by other run control statements and any valid requests. A FILE REPn statement must be coded for each additional report file (M4REP2-9). Refer to the VISION:Builder ASL Reference Guide for coding rules.
9	This statement calls in and executes the system sort program.
10	VISION:Builder is invoked to generate the report.
11	This M4LIST is output from a report step and includes a listing of run control statements, report messages, and the reports.
12	Defines the location of the sorted report file (M4REPI) which is the input to the report step.
13	This M4INPUT indicates to VISION:Builder that only the report step has to be executed.

Report from Master File and Coordinated Files (Three-Step)

VISION:Builder is able to handle a number of input files (one master and up to nine additional coordinated files) in one run. A coordinated file is a read only input file defined to VISION:Builder. When used as input to a run along with a master file, records become optionally available to you for processing such that:

- Matched records may be processed.
- Unmatched records may be processed.
- All records may be processed.

An RF statement must be coded for each additional coordinated file and specifies the DTF/DDname for the cord file, an alternate key field (if other than the key field in the file definition), and the type of coordination wanted.

[Figure 2-8 on page 2-23](#) and [Figure 2-9 on page 2-24](#) show the flow and JCL for a report from a master file and coordinated files (three-step). For a detailed explanation of coordinated files and types of coordination see the [VISION:Builder Reference Guide](#).

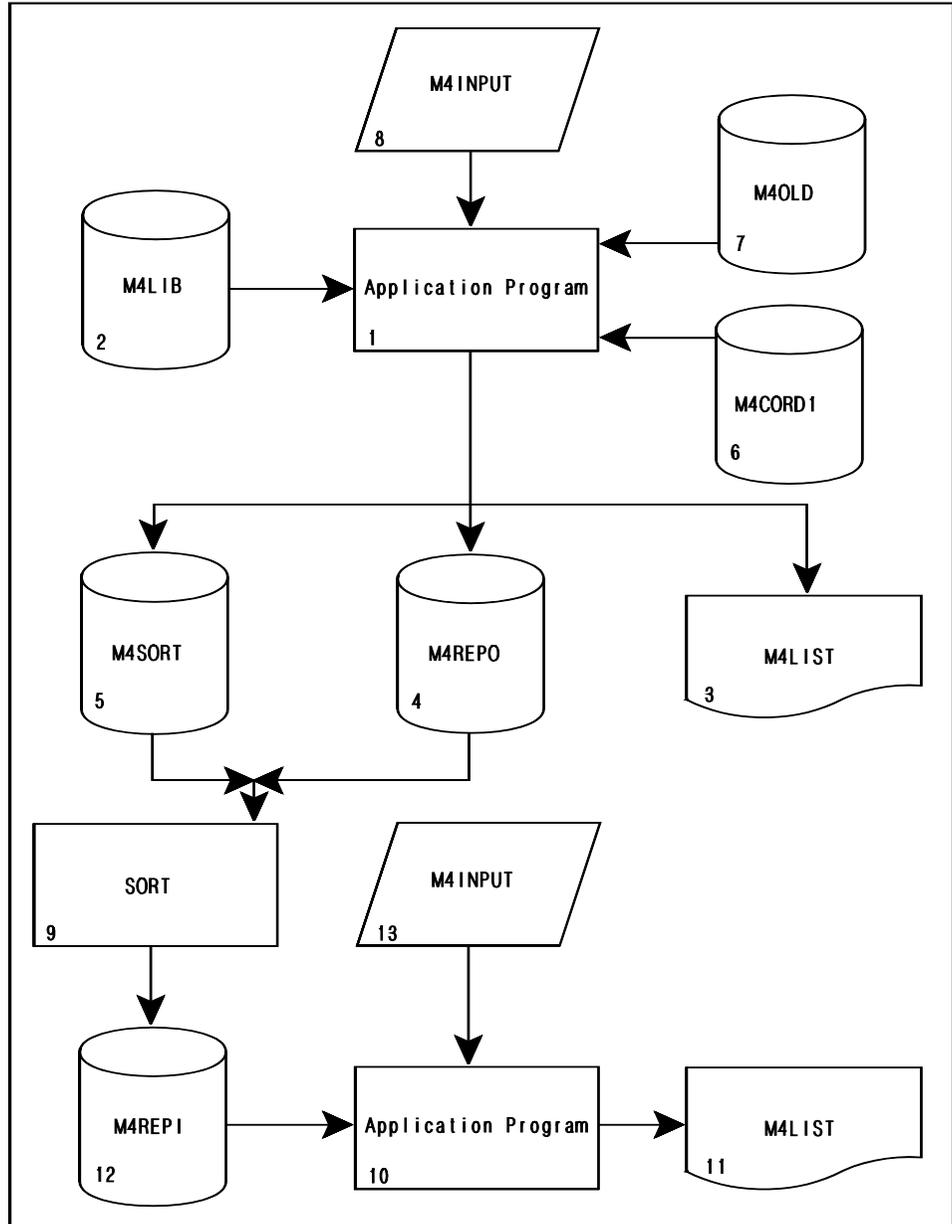


Figure 2-8 Flow for a Three-Step (Process-Sort-Report) Application Run with Coordinated File(s)

Notes

```

//          JOB (accounting information)
//* JCL FOR A THREE-STEP, WITH COORDINATED FILE **
//JOBLIB   DD DSN=your.builder.loadlib,DISP=SHR
1 //step    EXEC PGM=MARKIV,REGION=1536K
2 //M4LIB   DD DSN=your.m4lib,DISP=SHR
3 //M4LIST  DD SYSOUT=a
4 //M4REPO  DD DSN=your.m4repo,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,n)
5 //M4SORT  DD DSN=sort.file,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,n)
6 //M4CORD1 DD DSN=cord.file,DISP=(OLD,KEEP)
7 //M4OLD   DD DSN=old.master.file,DISP=(OLD,KEEP)
8 //M4INPUT DD *
CONTROL SORT EXTERNAL
FILE MASTER INPUT, NAME . . .
FILE CORD1 NAME . . .
FILE REPORT
:
:
/*
9 //sort    EXEC PGM=SORT
//SYSIN    DD DSN=sort.file,DISP=(OLD,PASS)
//SORTIN   DD DSN=your.m4repo,DISP=(OLD,DELETE)
//SORTOUT  DD DSN=your.m4repi,DISP=(NEW,PASS),
//          UNIT=sysda,SPACE=(TRK,(n,n))
//SYSOUT   DD SYSOUT=a
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
/*
10 //report  EXEC PGM=MARKIV,REGION=1536K
11 //M4LIST  DD SYSOUT=a
12 //M4REPI  DD DSN=your.m4repi,DISP=(OLD,DELETE)
13 //M4INPUT DD *
CONTROL
FILE REPORT
/*
//

```

Figure 2-9 JCL for a Three-Step (Process-Sort-Report) Application Run with Coordinated File(s)

The numbered statements in [Figure 2-9](#) represent those dealing with coordinated files. An explanation of the numbered statements follows:

Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors, and generates the necessary sort control statements. The execution of VISION:Builder is controlled by the run control statements, which specify the files and DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements and diagnostic messages.
4	M4REPO is the file containing the VISION:Builder report file output from the processing step. The data is used in subsequent sort and report steps.
5	The M4SORT file contains the sort control statements for the sort program.
6	M4CORD1 is the input file that is matched against the master file.
7	M4OLD defines the old master file and is the primary data input file to the processing step in this run.
8	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements. The FILE REPORT statement specifies that a report file is generated. The FILE CORD1 statement specifies an additional input file, M4CORD1. The run control statements are followed by any valid requests.
9	This statement calls in and executes the system sort program.
10	VISION:Builder is invoked to generate the report.
11	This M4LIST is output from a report run and includes a listing of the run control statements, report messages, and the reports.
12	M4REPI defines the location of the sorted report file output by the sort step, that is the input to the report step.
13	This M4INPUT indicates to VISION:Builder that only the report step has to be executed.

Create a Subfile and Generate a Report on an Alternate List File (Three-Step)

Subfiles can be created by outputting selected records from an input data set or by outputting selected fields. The [VISION:Builder ASL Reference Guide](#) shows how you can use your source statements to create a subfile.

Typically, the VISION:Builder source listing and report(s) are output to the M4LIST DD (generally the system printer). However, the use of alternate M4LISTs allows you to separate your output report(s) from the source listing and direct the report(s) to other destinations or devices.

The following methods are available for using alternate M4LISTs.

- Direct all your application's reports to an M4LIST1 DD by including LISTCNTL ALTLIST YES statement in your run control statements. The DD for M4LIST1 is specified in the report step of your application run JCL.

[Figure 2-10 on page 2-27](#) and [Figure 2-11 on page 2-28](#) show the flow and JCL for a three-step application run using an alternate list and a subfile where all reports are being directed to M4LIST1.

- Separate the reports individually by specifying your own specific ddname by including a FORMAT command with METHOD ALTLIST and DDNAME keywords for each report. See the [VISION:Builder ASL Reference Guide](#) for details. You must also provide JCL DD statements in the report step for each ddname specified on the En statements of your application.

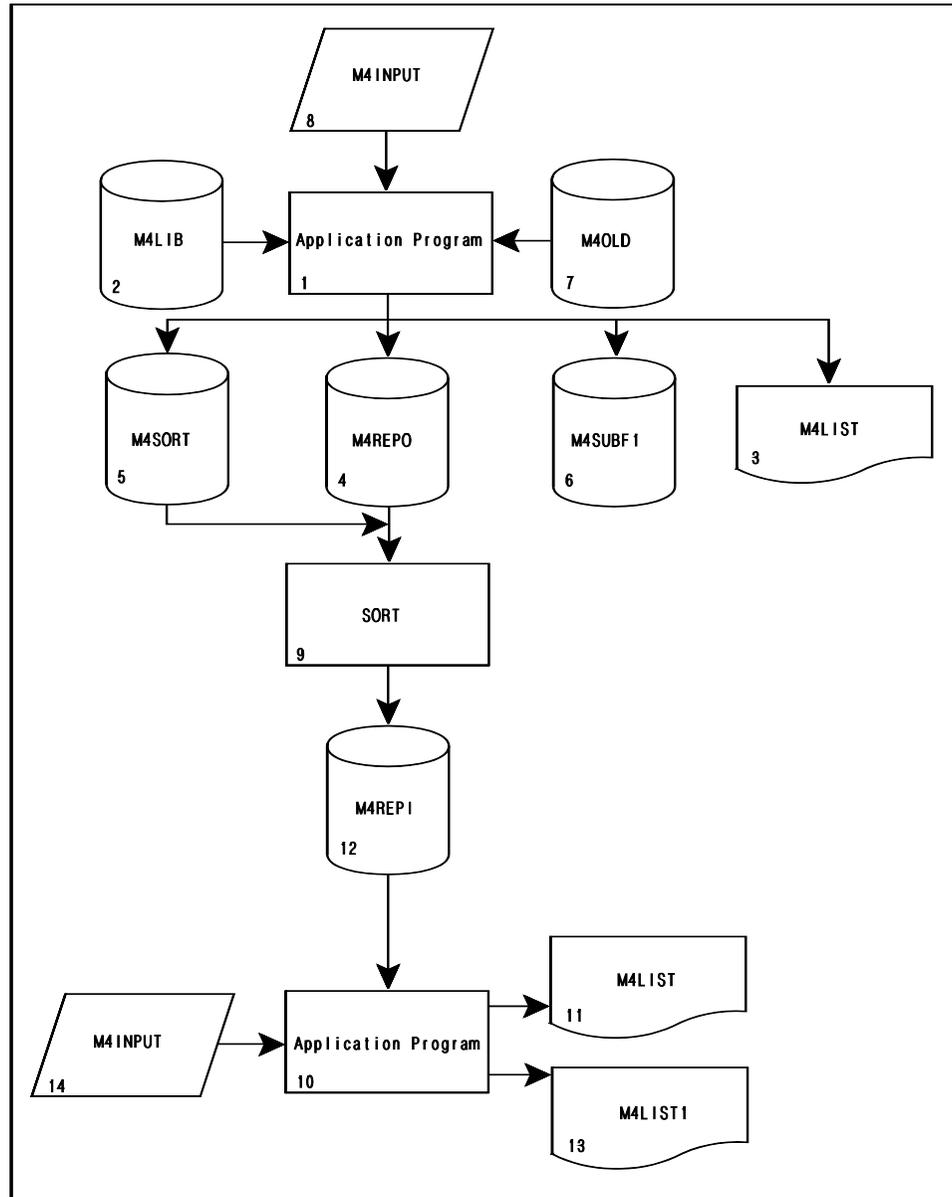


Figure 2-10 Flow for a Three-Step (Process-Sort-Report) Application Run Using a Subfile and M4LIST1

Notes

```

//          JOB (accounting information)
//* JCL FOR A THREE-STEP RUN WITH SUBFILES **
//JOB LIB DD DSN=your.builder.loadlib,DISP=(SHR,PASS)
1 //step EXEC PGM=MARKIV,REGION=1536K
2 //M4LIB DD DSN=your.m4lib,DISP=SHR
3 //M4LIST DD SYSOUT=a
4 //M4REPO DD DSN=your.m4repo,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,(n,n))
5 //M4SORT DD DSN=sort.file,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,n)
6 //M4SUBF1 DD DSN=sub.file,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,(n,n))
7 //M4OLD DD DSN=old.master.file,DISP=(OLD,KEEP)
8 //M4INPUT DD *
CONTROL SORT EXTERNAL
LISTCNTL ALTLIST YES
FILE MASTER INPUT, NAME . . .
FILE SUBF1 NAME . . .
:
:
/*
9 //sort EXEC PGM=SORT
//SYSIN DD DSN=sort.file,DISP=(OLD,PASS)
//SORTIN DD DSN=your.m4repo,DISP=(OLD,DELETE)
//SORTOUT DD DSN=your.m4repi,DISP=(NEW,PASS),
//          UNIT=sysda,SPACE=(TRK,(n,n))
//SYSOUT DD SYSOUT=a
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
/*
10 //report EXEC PGM=MARKIV,REGION=1536K
11 //M4LIST DD SYSOUT=a
12 //M4REPI DD DSN=your.m4repi,DISP=(OLD,DELETE)
13 //M4LIST1 DD DSN=builder.list,DISP=(NEW,CATLG),UNIT=sysda,
//          SPACE=(TRK,(n,n))
14 //M4INPUT DD *
CONTROL
FILE REPORT
/*
//

```

Figure 2-11 JCL for a Three-Step (Process-Sort-Report) Application Run Using a Subfile and M4LIST1

The numbered statements in [Figure 2-11](#) represent those dealing with subfiles and M4LIST1. An explanation of the numbered statements follows:

Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors, and generates the necessary sort control statements. The execution of VISION:Builder is controlled by run control statements that specify the files and DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.

Notes	Explanation
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements and diagnostic messages.
4	M4REPO is the file containing the VISION:Builder report file output from the processing step. The data is used in subsequent sort and report steps.
5	The M4SORT file contains the sort control statements for the sort program.
6	Describes the file (M4SUBF1) which will hold the subfile data.
7	M4OLD defines the old master file and is the primary data input file to the processing step in this run.
8	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements. The FILE SUBF1 statement is coded for the subfile (M4SUBF1). Refer to the VISION:Builder ASL Reference Guide for coding rules. The run control statements are followed by any valid requests.
9	This statement calls in and executes the system sort program.
10	VISION:Builder is invoked to generate the report.
11	This M4LIST is output from a report run and includes a listing of the source statements and report messages.
12	Defines the location of the sorted report file (M4REPI), output by the VISION:Builder processing run, that is the input to the report step.
13	An alternate M4LIST file containing reports and report messages for that file. The LISTCNTL ALTLIST YES statement in the processing step causes all reports to be output to M4LIST1. Note: If you had specified the alternate list ddnames in your En statements, you would provide a DD for each name specified in place of the M4LIST1 DD statement.
14	This M4INPUT indicates to VISION:Builder that only the report step has to be executed.

Update a Master File (Single-Step Sort) ⁴

Updating the master file is achieved through a transaction processing step. A transaction processing step inputs a transaction file (M4TRAN) composed of transaction records that are applied against the old master file (M4OLD). A detailed discussion of transaction processing is found in the [VISION:Builder Reference Guide](#).

During updating, you can output a file of deleted master file records (M4AUDIT) or rejected transaction records (M4REJCT) by coding the FILE AUDIT or FILE REJECT run control statements and also including the JCL statements for each file. Refer to the [VISION:Builder ASL Reference Guide](#) for coding rules.

After processing, if the record is not deleted, it can be output to a new master file (M4NEW), which is the updated version of the master file. It can also be used for update-in-place with the old master file.

[Figure 2-12](#) and [Figure 2-13](#) show the flow and JCL for an updated master file run (single-step sort).

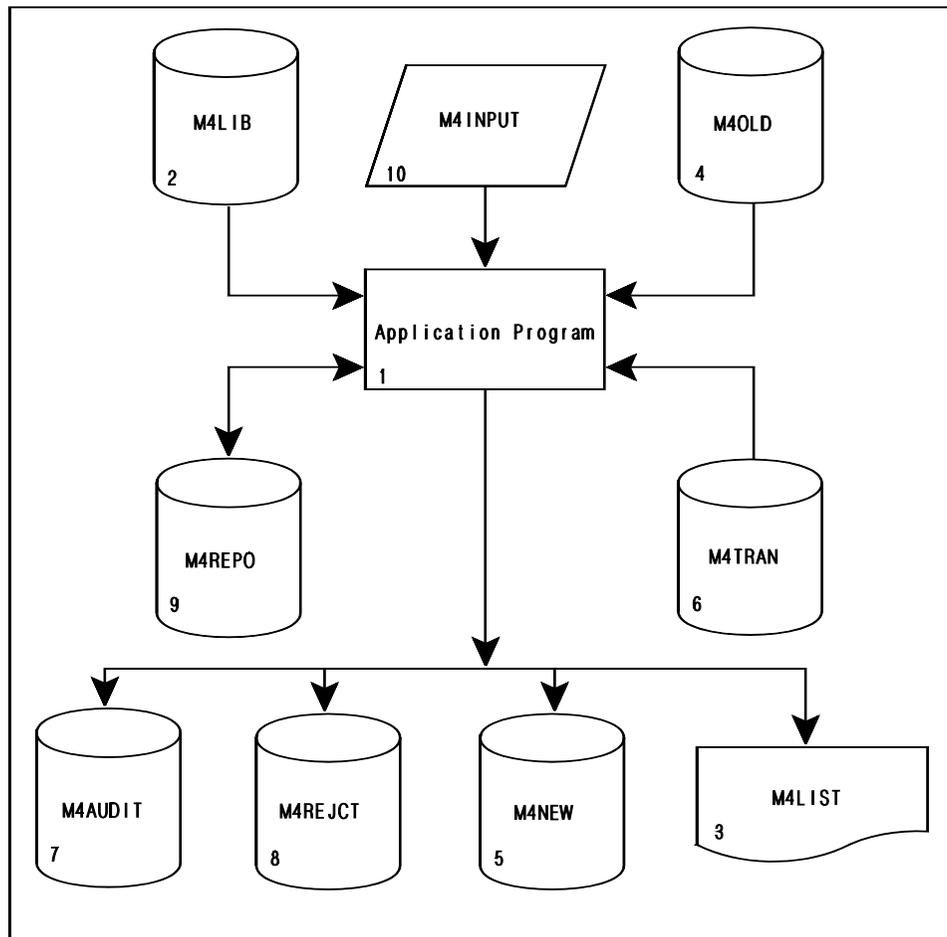


Figure 2-12 Flow and JCL for a Single-Step Application Run (Sort)

Notes

```

//          JOB    (accounting information)
** JCL FOR AN UPDATE MASTER FILE USING AUDIT AND REJECT FILES **
//JOBLIB    DD    DSN=your.builder.loadlib,DISP=SHR
1 //step     EXEC  PGM=MARKIV,REGION=1536K
2 //M4LIB   DD    DSN=your.m4lib,DISP=SHR
3 //M4LIST  DD    SYSOUT=a
4 //M4OLD   DD    DSN=old.master.file,DISP=(OLD,KEEP)
5 //M4NEW   DD    DSN=new.file,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,(n,n))
6 //M4TRAN  DD    DSN=tran.file,DISP=SHR
7 //M4AUDIT DD    DSN=audit.file,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,(n,n))
8 //M4REJCT DD    DSN=reject.file,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,(n,n))
//SYSOUT   DD    SYSOUT=a
//SORTWK01 DD    UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD    UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD    UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTLIB  DD    DSN=SYS1.SORTLIB,DISP=SHR
9 //M4REPO  DD    DSN=your.m4repo,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,(n,n))
10 //M4INPUT DD    *
CONTROL SORTSIZE 150K, REPTSIZE 150K
FILE MASTER INPUT, NAME . . .
FILE MASTER OUTPUT
FILE TRAN
FILE AUDIT
FILE REJECT
FILE REPORT
.
.
/*
//

```

Figure 2-13 JCL for a Single-Step (Sort) Application Run Using Audit, Transaction, and Reject Files and New Master Files

The numbered statements in [Figure 2-13](#) represent those dealing with updating a master file, M4AUDIT, M4REJCT, and M4TRAN files. An explanation of the numbered statements follows:

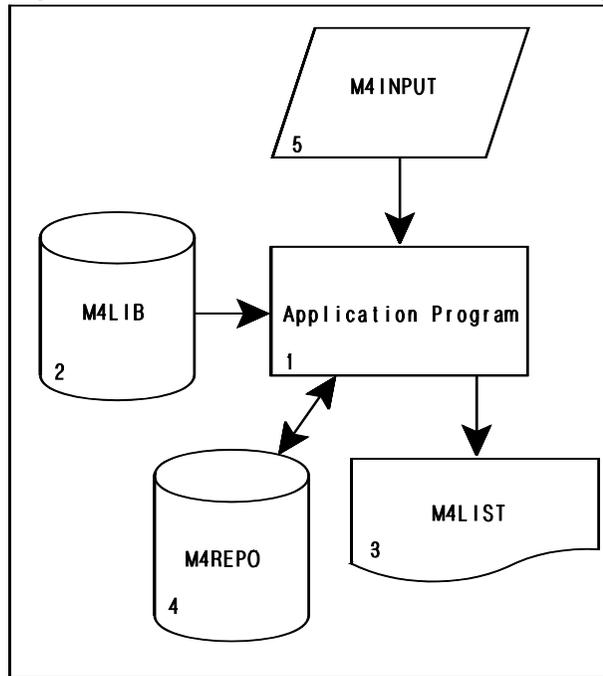
Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors, and generates the necessary sort control statements. The execution of VISION:Builder is controlled by run control statements that specify the files and DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements, diagnostic messages, and the report(s).

Notes	Explanation
4	M4OLD defines the old master file and is the primary data input file to the processing step in this run.
5	This statement defines the new master file (M4NEW).
6	This statement defines the transaction file (M4TRAN) used to process against the master file.
7	This statement defines the file (M4AUDIT) that holds the deleted master file records.
8	This statement defines the file (M4REJECT) that holds the rejected transaction records.
9	M4REPO is the file containing the VISION:Builder report command language records output during the processing phase. The report data records are passed to the sort processes.
10	<p>The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements.</p> <p>The SORTSIZE and REPTSIZE keywords may need to be coded when the default values specified in M4PARAMS is not adequate.</p> <ul style="list-style-type: none">■ SORTSIZE indicates the amount of main storage VISION:Builder allocates for the sort.■ REPTSIZE indicates the amount of storage VISION:Builder allocates for the report generation process.

Scan/Sample Report

A scan/sample report is produced by coding the SAMPLE keyword on the CONTROL statement. This causes VISION:Builder to decode requests without doing any processing. No input data file (M4OLD) is necessary in your JCL. Diagnostic messages and a sample report (if requested) for each valid request are produced allowing you to check the appearance of your data.

Figure 2-14 shows the flow and JCL for a scan/sample report (single-step no-sort).



Notes

```

//      JOB (accounting information)
//* JCL FOR A SCAN SAMPLE RUN **
//JOBLIB DD DSN=your.builder.loadlib,DISP=SHR
1 //scan EXEC PGM=MARKIV,REGION=1536K
2 //M4LIB DD DSN=your.m4lib,DISP=SHR
3 //M4LIST DD SYSOUT=a
4 //M4REPO DD DSN=your.m4repo,DISP=(NEW,DELETE),UNIT=sysda,
//      SPACE=(TRK,(n,n))
5 //M4INPUT DD *
CONTROL SORT NONE, SAMPLE
FILE MASTER INPUT, NAME . . .
FILE REPORT
.
.
/*
//

```

Figure 2-14 Flow and JCL for a Scan/Sample Report Single-Step (No-Sort) Run

The numbered statements in [Figure 2-14](#) represent those dealing with a scan/sample report. An explanation of the numbered statements follows:

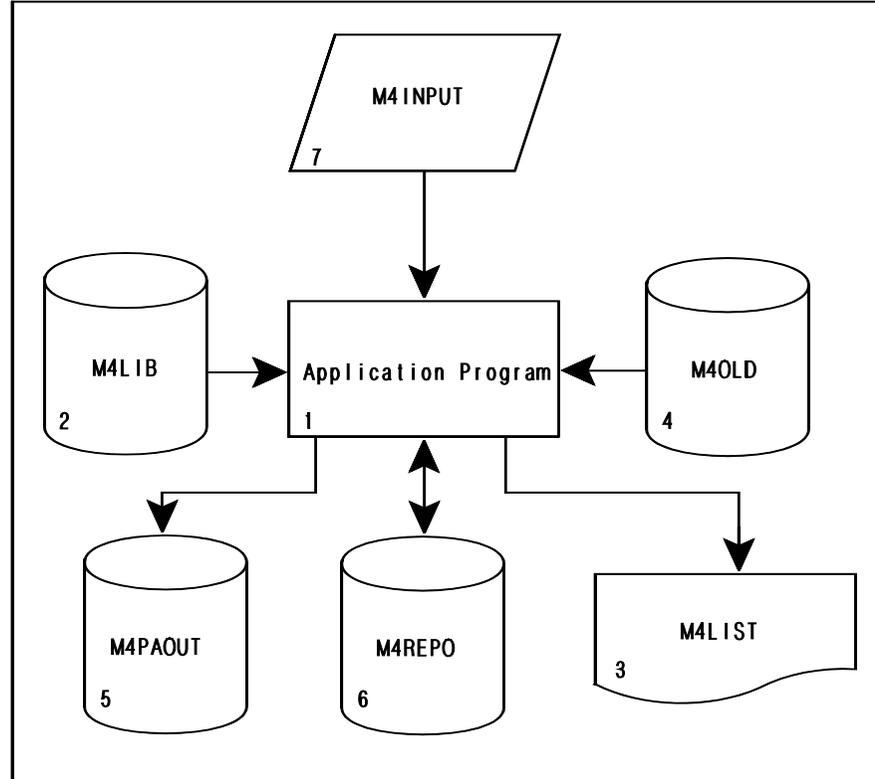
Notes	Explanation
1	VISION:Builder decodes the source statements from the M4INPUT data set, checking for syntactical errors. The execution of VISION:Builder is controlled by run control statements that specify the files and DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements, diagnostic messages, and the sample report(s).
4	The M4REPO statement defines the file containing the VISION:Builder report command language records.
5	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements and is followed by other run control statements and any valid requests.

Program Analyzer

The program analyzer (PA) statement is an optional part of the run control group that provides the user with various means to enhance program documentation and assist in the debugging process.

The PA statement options allow you to enhance the source statement listings so that they identify the VISION:Builder automatic numeric conversions, as well as create a VISION:Builder file, M4PAOUT. This file enables you to create two reports: Cross Reference and Execution Trace. Refer to the [VISION:Builder Reference Guide](#) for a detailed discussion of the Program Analyzer.

[Figure 2-15 on page 2-35](#) shows the flow and JCL for a single-step sort run using the program analyzer for execution trace reports and numeric conversions. M4PAOUT represents a file that is created during a processing step and brought back into a separate report step as M4OLD. [Figure 2-15 on page 2-35](#) also shows the necessary JCL to create the M4PAOUT file. [Figure 2-16 on page 2-37](#) shows the flow and JCL for a single-step (sort) run using M4PAOUT to produce the trace reports. [Figure 2-17 on page 2-39](#) shows the flow and JCL for a three-step run using the program analyzer cross reference feature.



Notes

```

1 // JOB
2 // * JCL FOR PROGRAM ANALYZER **
3 // JOBLIB DD DSN=your.builder.loadlib,DISP=SHR
4 // app EXEC PGM=MARKIV,REGION=1536K
5 // M4LIB DD DSN=your.m4lib,DISP=SHR
6 // M4LIST DD SYSOUT=a
7 // M4OLD DD DSN=old.master.file,DISP=(OLD,KEEP)
8 // M4PAOUT DD DSN=m4paout.file,DISP=(NEW,CATLG),
// UNIT=sysda,
// SPACE=(TRK,(n,n))
// SYSPRINT DD SYSOUT=a
// SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,CONTIG)
// SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,CONTIG)
// SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,CONTIG)
// SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
6 // M4REPO DD DSN=your.m4repo,DISP=(NEW,PASS),
// UNIT=sysda,SPACE=(TRK,(n,n))
7 // M4INPUT DD *
8 CONTROL SORTSIZE 200K,FREESIZE 200K
DOCUMENT CONVMSGS,EXECTRACE
FILE MASTER INPUT,NAME . . .
FILE REPORT
.
.
/*
//

```

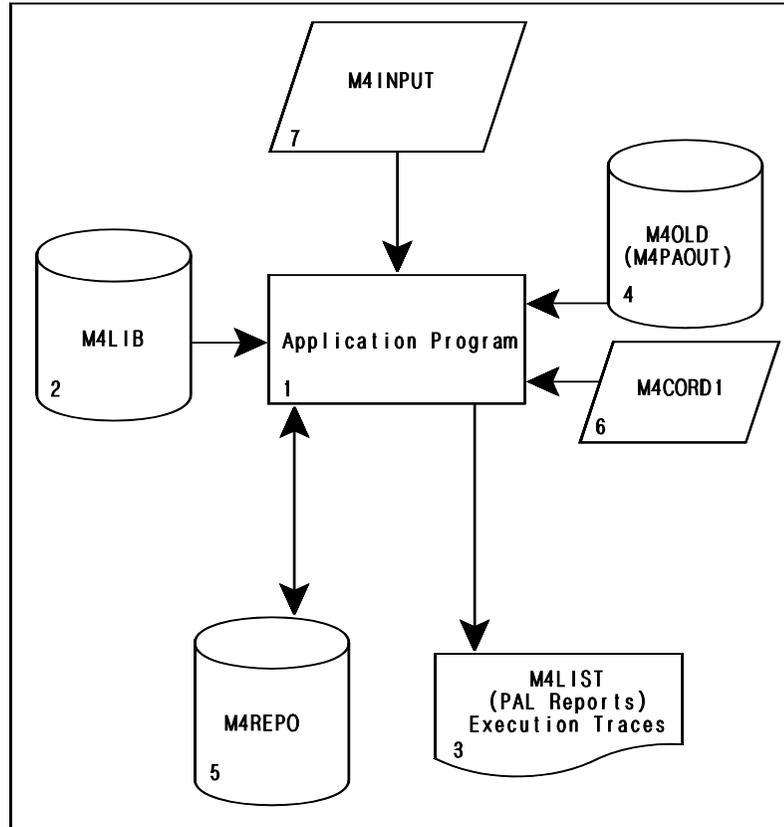
Figure 2-15 Flow and JCL for a Single-Step (Sort) Run, Creating the Program Analyzer Output File M4PAOUT

The numbered statements in [Figure 2-15](#) represent those dealing with creating the PAL output file, M4PAOUT. An explanation of the numbered statements follows:

Note: If you select Execution Trace only, you do not need to sort M4PAOUT before processing it.

Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors. The execution of VISION:Builder is controlled by run control statements that specify the files and DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements, arithmetic conversion messages, diagnostic messages, and the reports.
4	M4OLD defines the old master file and is the primary data input file to the processing step in this run
5	The M4PAOUT statement represents the output file for the program analyzer (PAL).
6	M4REPO is the file containing the VISION:Builder report command language records output from the processing phase. The data is then used in report generation.
7	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements. The CONTROL statement is followed by other run control statements and any valid requests.
8	The DOCUMENT statement invokes the program analyzer. The CONVMSG keyword causes numeric messages to be printed. The EXECTRACE keyword causes trace data to be output to M4PAOUT.

The PAL output file, M4PAOUT, was created in [Figure 2-15](#). [Figure 2-16](#) shows the flow and JCL for processing M4PAOUT as M4OLD.



Notes

```

1 // JOB
2 // * JCL USING M4PAOUT **
3 // JOBLIB DD DSN=your.builder.loadlib,DISP=SHR
4 // step EXEC PGM=MARKIV,REGION=1536K
5 // M4LIB DD DSN=your.m4lib,DISP=SHR
6 // M4LIST DD SYSOUT=a
7 // M4OLD DD DSN=m4paout.file,DISP=SHR
// SYSOUT DD SYSOUT=a
// SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
// SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
// SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
5 // M4REPO DD DSN=your.m4repo,DISP=(NEW,PASS),UNIT=synda,
6 // SPACE=(TRK,(n,n))
// M4CORD1 DD *
IGCPAL YY
/*
7 //M4INPUT DD *
CONTROL DELIMITER '#', SORTSIZE 200K
FILE MASTER INPUT, NAME IGCPALVB, KEYS NONE
FILE CORD1 NAME IGCPALRS, USER_READ
FILE REPORT
WORK AREA 1, NAME IGCPALWK
;
INCLUDE IGCPAL
/*
//
  
```

Figure 2-16 Flow and JCL for a Single-Step (Sort) Run, Using M4PAOUT to Produce Reports (Execution Trace)

The numbered statements in [Figure 2-16](#) represent those dealing with the use of the PAL output file, M4PAOUT. An explanation of the numbered statements follows:

Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors. The execution of VISION:Builder is controlled by run control statements, which specify the files and DD statements required for this job.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the PAL file definitions and cataloged requests.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements, diagnostic messages, and the execution trace reports.
4	During this run M4OLD represents the M4PAOUT file created in Figure 2-15 on page 2-35 . It is the output file for the program analyzer that will enable you to create PAL cross reference or execution trace reports. In this example, it produces execution trace reports.
5	M4REPO is the file containing the VISION:Builder command language records used to create the reports.
6	The M4CORD1 data set consists of a control card specifying the literal IGCPAL in positions 1-6 and Ys in appropriate positions for requesting specific reports (refer to the VISION:Builder Reference Guide). In this example, only the request execution trace reports are generated.
7	<p>The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements.</p> <p>Note: The system delimiter must be pound sign (#) and that this is a single-step run (default to SORT INTERNAL).</p> <ul style="list-style-type: none"> ■ The FILE MASTER statement with name IGCPALVB specifies the PAL master file. ■ The FILE CORD1 statement supplies the request-read coordinated file (M4CORD1) information, including the PAL definition name for the coordinated file. ■ The WORK AREA statement specifies the working storage file used by the PAL cataloged requests. ■ The INCLUDE statement is used to bring in the required request group.

Figure 2-17 shows the flow and JCL for a three-step application run using the program analyzer cross reference feature. Note that when cross reference reports are to be generated, the M4PAOUT file must be sorted before using it as M4OLD to generate the reports

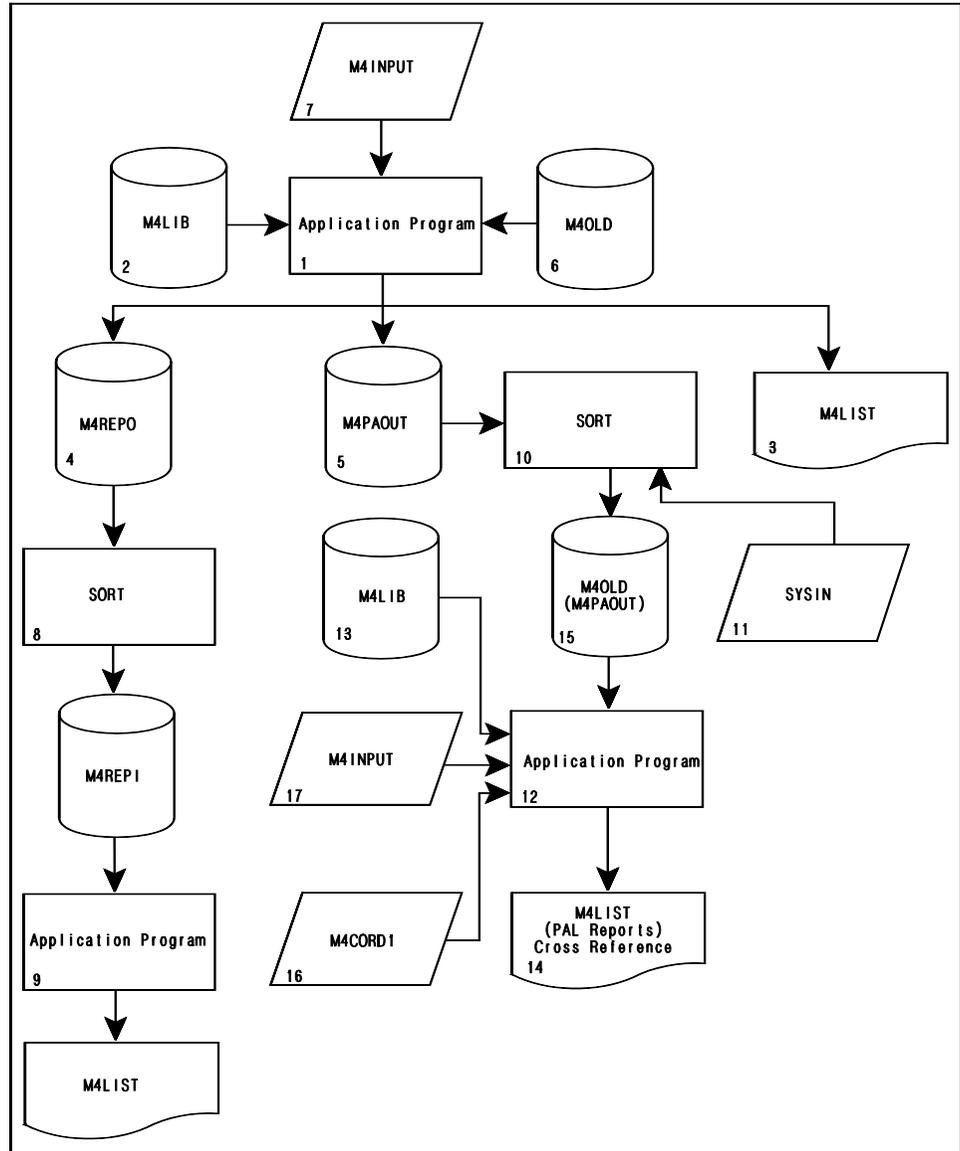


Figure 2-17 Flow for a Three-Step Application Run (Process-Sort-Report) Using the Program Analyzer Cross-Reference Feature

Note: In Figure 2-17, for SYSIN, the sort control statements are shown in Figure 2-18 on page 2-40.

```

Notes
//      JOB (accounting information)
/** JCL FOR A THREE-STEP APPLICATION RUN **
//JOBLIB DD DSN=your.builder.loadlib,DISP=SHR
1 //steпа EXEC PGM=MARKIV,REGION=1536K
2 //MALIB DD DSN=your.m4lib,DISP=SHR
3 //MALIST DD SYSOUT=a
4 //MAREPO DD DSN=your.m4repo,DISP=(NEW,PASS),UNIT=sysda,
//      SPACE=(TRK,(n,n),RLSE)
//MASORT DD DSN=%%SORT,DISP=(NEW,PASS,DELETE),
//      SPACE=(TRK,1),UNIT=SYSDA
5 //MAPAOUT DD DSN=%%PAOUT,DISP=(NEW,PASS),
//      UNIT=sysda,
//      SPACE=(TRK,(n,n),RLSE)
6 //MAOLD DD DSN=old.master.file,DISP=(OLD,KEEP)
7 //MAINPUT DD *
CONTROL SORT EXTERNAL
DOCUMENT XREF
FILE MASTER INPUT, NAME . . .
FILE REPORT
.
.
.
8 //sort EXEC PGM=SORT,REGION=1536K
//SYSIN DD DSN=%%SORT,DISP=(OLD,DELETE)
//SORTIN DD DSN=your.m4repo,DISP=(OLD,DELETE)
//SORTOUT DD DSN=%%MAREPI,DISP=(NEW,PASS),
//      SPACE=(TRK,(n,n),RLSE),UNIT=SYSDA
//SYSOUT DD SYSOUT=A
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
9 //report EXEC PGM=MARKIV,REGION=1536K
//MAREPI DD DSN=%%MAREPT,DISP=(OLD,DELETE)
//MAINPUT DD *
CONTROL
FILE REPORT
/**
//MALIST DD SYSOUT=A
10 //sort EXEC PGM=MARKIV,REGION=1536K
11 //SYSIN DD *
SORT FIELDS=(9,50,A),FORMAT=BI
RECORD TYPE=V,LENGTH=(168,168,168,65,168)
END
//SORTIN DD DSN=%%PAOUT,DISP=(,PASS)
//SORTOUT DD DSN=%%PAREPT,DISP=(NEW,PASS),
//      UNIT=sysda,
//      SPACE=(TRK,(n,n),RLSE)
//SYSOUT DD SYSOUT=a
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
12 //usepal EXEC PGM=MARKIV,REGION=1536K
13 //MALIB DD DSN=your.m4lib,DISP=SHR
14 //MALIST DD SYSOUT=a
15 //MAOLD DD DSN=%%PAREPT,DISP=(,PASS)
16 //MACORD1 DD *
IGCPAL YYYY
/**
//SYSOUT DD SYSOUT=a
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//MAREPO DD DSN=your.m4repo,DISP=(NEW,DELETE),UNIT=sysda,
//      SPACE=(TRK,(n,n),RLSE)
17 //MAINPUT DD *
CONTROL DELIMITER '#', SORTSIZE 200K
FILE MASTER INPUT, NAME IGCPALWB, KEYS NONE
FILE CORD1 NAME IGCPALRS, USER_READ
FILE REPORT
WORK AREA 1, NAME IGCPALWK
;
INCLUDE IGCPAL
/**
//

```

Figure 2-18 JCL for a Three-Step Application Run (Process-Sort-Report) Using the Program Analyzer Cross Reference Feature

An explanation of the numbered statements for [Figure 2-18](#) follows:

Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors, and generates the necessary sort control statements. The execution of VISION:Builder is controlled by run control statements, which specifies the files required, and the DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements and diagnostic messages.
4	M4REPO is the file containing the VISION:Builder report file output from the processing step. The data is used in subsequent sort and report steps.
5	M4PAOUT is the output file for the program analyzer.
6	M4OLD defines the old master file and is the primary data input file to the processing step in this run.
7	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements. The CONTROL statement is followed by other run control statements and any valid requests. The DOCUMENT statement specifies that only cross reference data is to be output to the M4PAOUT file. Note that, in this run, the M4PAOUT file is a temporary data set.
8	A stand alone sort step is executed to sort the VISION:Builder report file produced during the processing step.
9	VISION:Builder is invoked to generate the VISION:Builder reports.
10	This sort step sorts the M4PAOUT file produced during the processing step.
11	This statement defines the sort control statements needed to sort the M4PAOUT file.
12	VISION:Builder is invoked to generate the PAL cross reference reports.
13	The common library defines the cataloged data set that contains the PAL definitions and requests.
14	This M4LIST statement defines the location of the system output device for this step and includes a listing of the run control statements, report messages, and the PAL cross reference report(s).

Notes	Explanation
15	The sort step returns a sorted M4PAOUT file (&&PAREPT) that is used as input (M4OLD) to the report step.
16	The request-read coordinated file contains a user supplied record specifying which PAL reports are to be generated. The literal IGCPAL must be in positions 1-6. In this example, all four cross reference reports are to be generated.
17	<p>The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements.</p> <p>Note: The system delimiter must be pound sign (#) and that this is a single-step run (default to SORT INTERVAL).</p> <ul style="list-style-type: none">■ The FILE MASTER statement with name IGCPALVB specifies the PAL master file.■ The FILE CORD1 statement supplies the request-read coordinated file (M4CORD1) information, including the PAL definition name for the coordinated file.■ The WORK AREA statement specifies the working storage file used by the PAL cataloged requests. <p>The INCLUDE statement is used to bring in the required request group.</p>

Report Summary File

A report summary file can be created during the report phase of VISION:Builder for a report that has summaries specified. You can choose to create a report summary file by specifying a ddname with the SUMFILE keyword on the FORMAT statement for the report (see the [VISION:Builder ASL Reference Guide](#)) and providing JCL for the report summary file. Up to 255 report summary files can be produced, one for each report.

[Figure 2-19](#) and [Figure 2-20](#) show the flow and JCL for creating two report summary files in a three-step run. [Figure 2-21 on page 2-46](#) and [Figure 2-22 on page 2-47](#) show the flow and JCL for a single-step run. If a report summary file is requested in a three-step sample report run, the JCL statement for the report summary file must appear in the decode step instead of the report step.

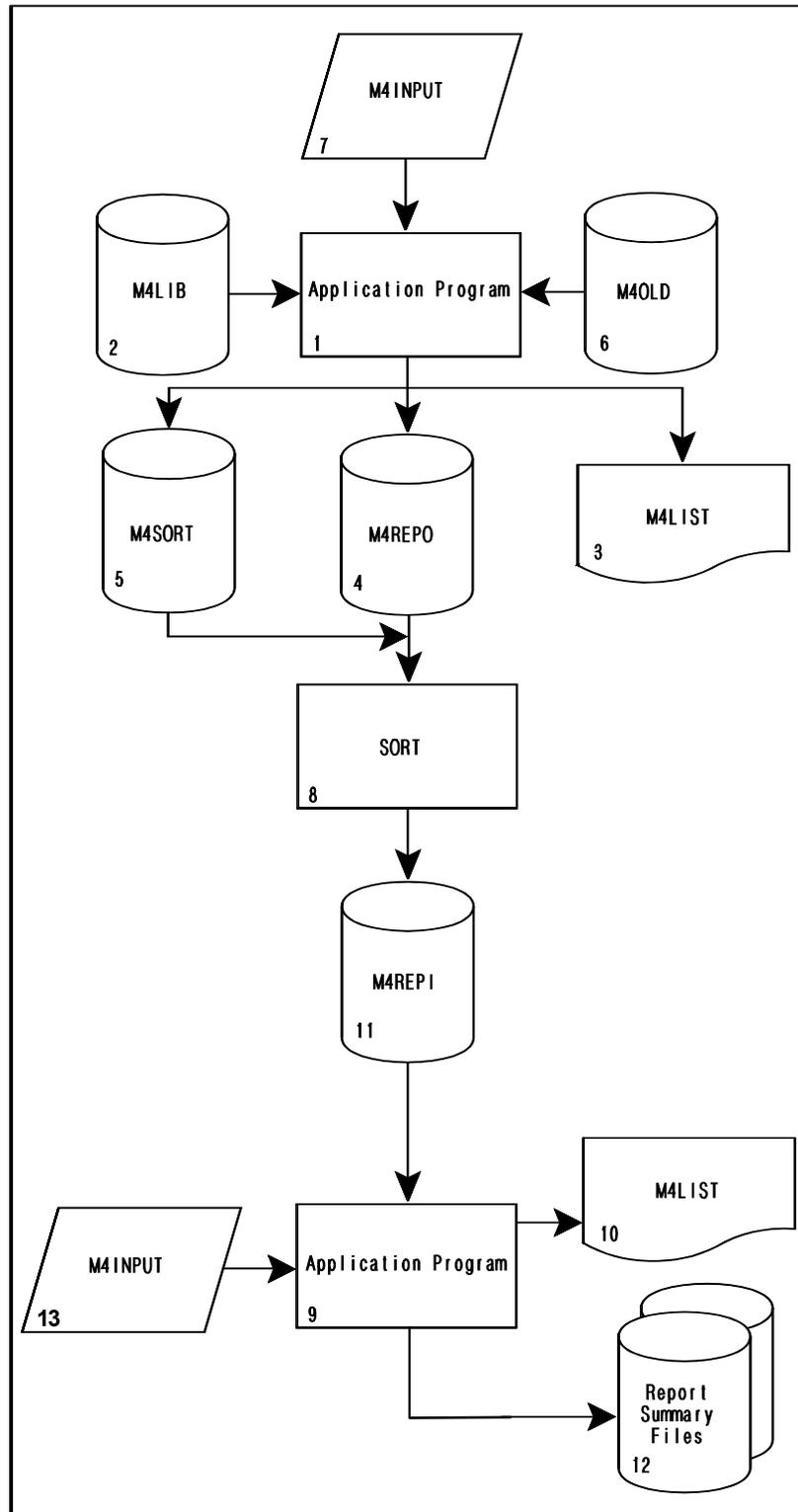


Figure 2-19 Flow for a Three-Step Application Run Using a Report Summary File

```

Notes
//      JOB (accounting information)
/** JCL FOR A THREE-STEP RUN WITH REPORT SUMMARY FILES **
//JOBLIB DD DSN=your.builder.loadlib,DISP=(SHR,PASS)
1 //step EXEC PGM=MARKIV,REGION=1536K
2 //MALIB DD DSN=your.m4lib,DISP=SHR
3 //MALIST DD SYSOUT=a
4 //MAREPO DD DSN=your.m4repo,DISP=(NEW,PASS),UNIT=sysda,
//      SPACE=(TRK,(n,n))
5 //MASORT DD DSN=sort.file,DISP=(NEW,PASS),UNIT=sysda,
//      SPACE=(TRK,n)
6 //MAOLD DD DSN=old.master.file,DISP=(OLD,KEEP)
7 //MAINPUT DD *
CONTROL SORT EXTERNAL
FILE MASTER INPUT, NAME . . .
FILE REPORT
.
.
.
REPORT . . .
  FORMAT SUMFILE RPTSUM01, . . .
END REPORT
;
REPORT . . .
  FORMAT SUMFILE RPTSUM02, . . .
END REPORT
8 /*
//sort EXEC PGM=SORT
//SYSIN DD DSN=sort.file,DISP=(OLD,PASS)
//SORTIN DD DSN=your.m4repo,DISP=(OLD,DELETE)
//SORTOUT DD DSN=your.m4repl,DISP=(NEW,PASS),
//      UNIT=sysda,SPACE=(TRK,(n,n))
//SYSOUT DD SYSOUT=a
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,CONTIG)
9 /*
10 //report EXEC PGM=MARKIV,REGION=1536K
11 //MALIST DD SYSOUT=a
12 //MAREPI DD DSN=your.m4repi,DISP=(OLD,DELETE)
//RPTSUM01 DD DSN=your.report.sumfile1,
//      DISP=(NEW,CATLG),UNIT=sysda,
//      SPACE=(TRK,(n,n))
12 //RPTSUM02 DD DSN=your.report.sumfile2,
//      DISP=(NEW,CATLG),UNIT=sysda,
//      SPACE=(TRK,(n,n))
13 //MAINPUT DD *
CONTROL
FILE REPORT
/*
//

```

Figure 2-20 JCL for a Three-Step Application Run Using a Report Summary File

The numbered statements in [Figure 2-20](#) represent those dealing with producing a report summary file. An explanation of the numbered statements follows:

Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors, and generates the necessary sort control statements. The execution of VISION:Builder is controlled by run control statements, which specify the files and DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements and diagnostic messages.
4	M4REPO is the file containing the VISION:Builder report file output from the processing step. The data is used in subsequent sort and report steps.
5	The M4SORT file contains the sort control statements for the sort program.
6	M4OLD defines the old master file and is the primary data input file to the processing step in this run.
7	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements. A FORMAT statement must be coded with a SUMFILE keyword for each report generating a summary file.
8	This statement calls in and executes the system sort program.
9	VISION:Builder is invoked to generate the report.
10	This M4LIST is output from a report run and includes a listing of the source statements and report messages.
11	Defines the location of the sorted report file (M4REPI), which is output by the VISION:Builder processing run and the input to the report step.
12	Two report summary files that contain control break fields and summary values for a report. RPTSUM01 contains the values from one report and RPTSUM02 from another report.
13	This M4INPUT indicates to VISION:Builder that only the report step has to be executed.

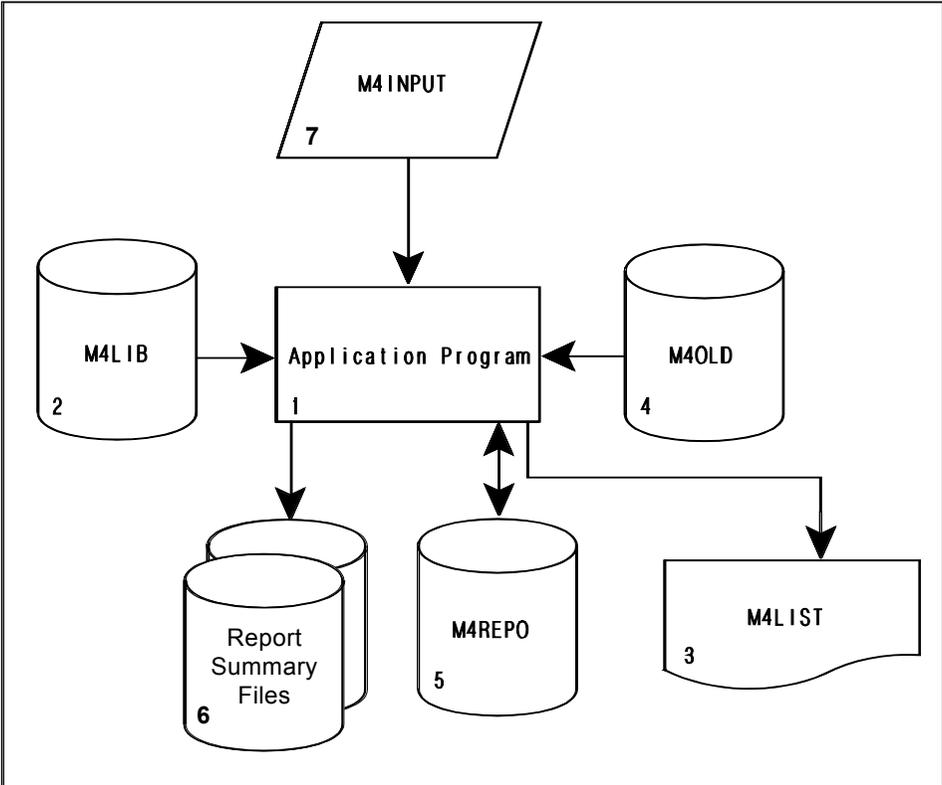


Figure 2-21 Flow for a Single-Step Run Using a Report Summary File

Notes

```

//          JOB (accounting information)
//* JCL FOR SINGLE-STEP RUN WITH A REPORT SUMMARY FILE **
//JOBLIB   DD DSN=your.builder.loadlib,DISP=SHR
1 //step    EXEC PGM=MARKIV,REGION=1536K
2 //M4LIB   DD DSN=your.m4lib,DISP=SHR
3 //M4LIST  DD SYSOUT=a
4 //M4OLD   DD DSN=old.master.file,DISP=(OLD,KEEP)
//SYSOUT   DD SYSOUT=a
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTLIB  DD DSN=SYS1.SORTLIB,DISP=SHR
5 //M4REPO  DD DSN=your.m4repo,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,(n,n))
6 //RPTSUM01 DD DSN=your.report.sumfile1,DISP=(NEW,CATLG),
//          UNIT=sysda,SPACE=(TRK,(n,n))
6 //RPTSUM02 DD DSN=your.report.sumfile2,DISP=(NEW,CATLG),
//          UNIT=sysda,SPACE=(TRK,(n,n))
7 //M4INPUT DD *
CONTROL SORTSIZE 200K
FILE MASTER INPUT, NAME . . .
FILE REPORT
.
.
REPORT . . .
  FORMAT SUMFILE RPTSUM01, . . .
END REPORT
;
REPORT . . .
  FORMAT SUMFILE RPTSUM02, . . .
END REPORT
/*
//

```

Figure 2-22 JCL for a Single-Step Run Using a Report Summary File

The numbered statements in [Figure 2-22](#) represent those dealing with producing a report summary file. An explanation of the numbered statements follows:

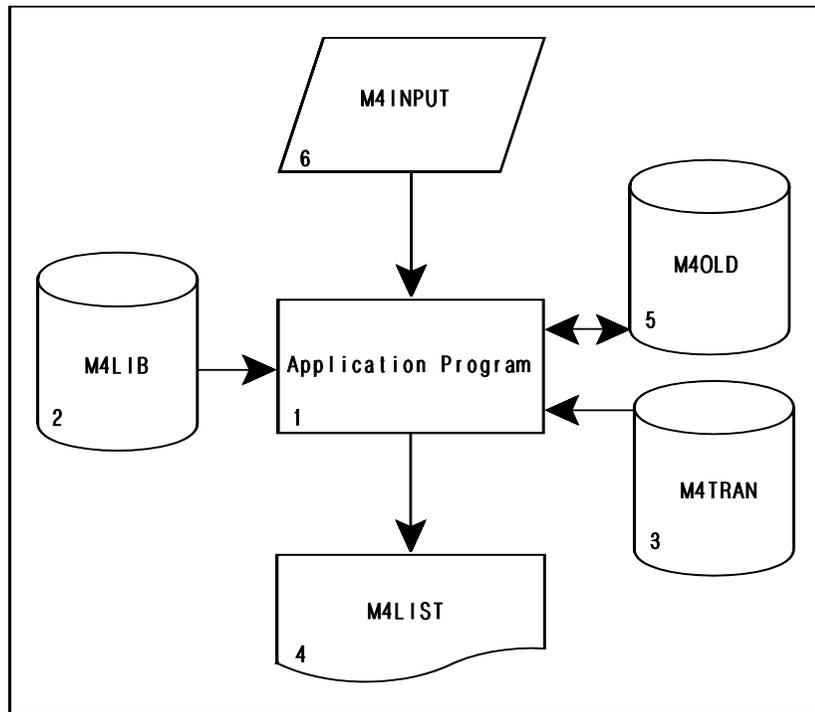
Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors, and generates the necessary sort control statements. The execution of VISION:Builder is controlled by run control statements that specify the files required and the DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements, diagnostic messages, and the report(s).
4	M4OLD defines the master file and is the primary data input to the processing step in this run.

Notes	Explanation
5	M4REPO is the file that will contain the report command language necessary to build the report from the processing step. The report data records are passed to the sort process.
6	These are report summary files that contain control break fields and summary values for a report. RPTSUM01 contains the values from one report and RPTSUM02 from another report.
7	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements. A single-step run requires that SORT INTERNAL be allowed as the default on the CONTROL statement. The CONTROL statement is followed by other run control statements and any valid requests. A FORMAT statement with the SUMFILE keyword must be coded for the report summary file.

Update-in-Place (Single-Step No-Sort)⁴

Updating a master file through transaction processing (see [Update a Master File \(Single-Step Sort\)](#) on page 2-29) can also be accomplished by performing an update-in-place. The input transaction file (M4TRAN) is composed of transaction records that are applied against the old master file (M4OLD). When using update-in-place, you do not need to specify a new master file, as an updated record occupies the same physical position on a direct-access device as the original old master record. Update-in-place allows you to update randomly accessed old master file records.

Figure 2-23 shows the flow and JCL for an update-in-place run.



Notes

```

//          JOB (accounting information)
//JOBLIB    DD DSN=your.builder.loadlib,DISP=SHR
1 //update  EXEC PGM=MARKIV,REGION=1536k
//M4LIB     DD DSN=your.m4lib,DISP=SHR
3 //M4TRAN   DD DSN=tran.file,DISP=SHR
4 //M4LIST   DD SYSOUT=a
//M4OLD     DD DSN=old.master.file,DISP=(OLD,KEEP)
5 //M4INPUT  DD *
6 CONTROL SORT NONE, REPTSIZE 200K
FILE MASTER UPDATE, NAME . . .
FILE TRAN
FILE REPORT
.
.
/*
//

```

Figure 2-23 Flow and JCL for an Update-In-Place, Single-Step (No-Sort) Application Run

Note: A master file can be updated-in-place only if the storage medium is a direct-access storage device.

The numbered statements in [Figure 2-23](#) represent those dealing with updating in place. An explanation of the numbered statements follows:

Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors. The execution of VISION:Builder is controlled by run control statements that specify the files required, and the DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.
3	M4TRAN defines the transaction record file.
4	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements, diagnostic messages, and the report.
5	M4OLD defines the old master file.
6	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements. The run control statements are followed by any valid requests.

Report Manager JCL Examples

The use of the Report Manager facility to COLLATE and/or ROUTE reports produced by VISION:Builder applications requires minor additions to the standard application run JCL. The following lists JCL for single-step and three-step application runs that use the Report Manager facility.

Report Manager Single-Step JCL for Collating and/or Routing

In single-step application runs, the Report Manager needs JCL statements that are the destinations for the collated and/or routed reports. The M4PRINT DD statement receives all the collated-only reports and any non-routed reports. Specifically named DD statements receive routed reports that correspond to the designated routing destinations. There are no other changes needed to the single-step application run JCL for Report Manager. Any Report Manager messages and run statistics are directed to M4LIST.

[Figure 2-24](#) shows a simple single-step JCL example with the additional JCL needed for Report Manager.

Notes

```

//          JOB (accounting information)
//*
//JOBLIB   DD DSN=your.builder.loadlib,DISP=SHR
//*
//* JCL for SINGLE-STEP PROCESSING RUN with REPORT MANAGER
//* (Process-Sort-Report)
//*
1 //step    EXEC PGM=MARKIV,REGION=1536K
2 //M4LIB   DD DSN=your.m4lib,DISP=SHR
3 //M4LIST  DD SYSOUT=*
4 //M4OLD   DD DSN=old.master.file,DISP=(OLD,KEEP)
//SYSOUT   DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
5 //SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
6 //M4REPO  DD DSN=your.m4repo,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,(n,n))
7 //M4INPUT DD *
CONTROL
FILE MASTER INPUT, NAME . . .
FILE REPORT
COLLATE REPORTS REP1 REP2, KEYLENGTH 1
ROUTE REPORT REP3 TO DEST001
;
REP1: REPORT . . .
. . .
END REPORT
;
REP2: REPORT . . .
. . .
END REPORT
;
REP3: REPORT . . .
. . .
END REPORT
/*
/* ADDITIONAL REPORT MANAGER JCL STATEMENTS
/*
8 //M4PRINT DD SYSOUT=*          COLLATE ONLY OUTPUT/NON-ROUTED OUTPUT
9 //DEST001 DD SYSOUT=*          ROUTE DESTINATION STATEMENTS AS NEEDED

```

Figure 2-24 Report Manager Single-Step Application Run JCL

An explanation of the numbered statements for [Figure 2-24](#) follows:

Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors, and generates the necessary sort control statements. The execution of VISION:Builder is controlled by run control statements that specify the files and DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements, diagnostic messages, and the run statistics.

Notes	Explanation
4	M4OLD defines the master file and is the primary data input to the processing step in this run.
5	SORTLIB defines the location of the system sort program.
6	M4REPO is the file containing the report command language and ROUTE commands necessary to build and route the reports. The report data records are passed to the sort process.
7	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements. As part of the run control statements for the application, the report manager COLLATE and/or ROUTE Commands are specified.
8	<p>The M4PRINT statement defines a location for Report Manager output.</p> <ul style="list-style-type: none"> ■ In collate-only runs, the collated reports are directed here first followed by the non-collated reports. ■ In runs that perform routing, the non-routed reports are directed here. ■ If the M4PRINT statement is present without any report manager commands in the application source code, all reports are directed here instead of M4LIST. <p>Note: The report output from VISION:Builder can be directed to SYSOUT, disk and tape data sets, PDS members, or wherever your operating system JCL specification will allow.</p>
9	<p>The DEST001 statement defines the destination for report REP3.</p> <p>Additional statements are needed for each designated destination name specified in the report manager ROUTE commands.</p>

Report Manager Three-Step JCL for Collating with or without Routing

In three-step application runs, minor JCL changes are needed for the Report Manager when performing collating with or without routing.

There are no changes needed in the processing step; however, your application source code will include your collating and routing specifications.

The sort step needs to execute the program called COLLATOR that performs a portion of the collating specifications while using the SORT program to sort the report file. An M4LIST DD statement is needed in the sort step for receiving any COLLATOR program messages.

Note: The COLLATOR program can only be used to sort report files from applications using COLLATE commands.

Finally, the report step needs JCL statements that are the destinations for the collated and/or routed reports. The M4PRINT DD statement receives all the collated-only reports and any non-routed reports. Specifically named DD statements receive routed reports that correspond to the designated routing destinations.

[Figure 2-25](#) shows a simple three-step JCL example with the additional JCL changes needed for the Report Manager.

Notes

```

//          JOB (accounting information)
//JOBLIB   DD DSN=your.builder.loadlib,DISP=SHR
//*
//* JCL for THREE-STEP APPLICATION RUN with REPORT MANAGER
//*
//* PROCESSING STEP - COLLATE Commands present
//*          (with or without ROUTE Commands)
//*
1 //stepa EXEC PGM=MARKIV,REGION=1536K
2 //MALIB DD DSN=your.m4lib,DISP=SHR
3 //MALIST DD SYSOUT=*
4 //MAOLD DD DSN=old.master.file,DISP=(OLD,KEEP)
5 //MASORT DD DSN=sort.file,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,n)
6 //MAREPO DD DSN=your.m4repo,DISP=(NEW,PASS),UNIT=sysda,
//          SPACE=(TRK,(n,n))
7 //MAINPUT DD *
CONTROL SORT EXTERNAL
FILE MASTER INPUT, NAME . . .
FILE REPORT
COLLATE REPORTS REP1 REP2, KEYLENGTH 1
ROUTE REPORT REP3 TO DEST001
;
REP1: REPORT . . .
. . .
END REPORT
;
REP2: REPORT . . .
. . .
END REPORT
;
REP3: REPORT . . .
. . .
END REPORT
/*
/* SORT STEP - The COLLATOR Program performs some COLLATE functions
/*
8 //stepb EXEC PGM=COLLATOR,REGION=1536K
/*
9 //SYSIN DD DSN=sort.file,DISP=(,PASS)
//SORTIN DD DSN=your.m4repo,DISP=(OLD,DELETE)
//SORTOUT DD DSN=your.m4repi,DISP=(NEW,PASS),
//          UNIT=sysda,SPACE=(TRK,(n,n))
//SYSOUT DD SYSOUT=*
//SORTLIB DD DSN=SYS1,SORTLIB,DISP=SHR
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
/*
/* ADDITIONAL REPORT MANAGER (COLLATOR) JCL STATEMENT
/*
10 //MALIST DD SYSOUT=*          COLLATOR Program message output
/*
/* REPORT STEP
/*
11 //stepc EXEC PGM=MARKIV,REGION=1536K
12 //MALIST DD SYSOUT=*
13 //MAREPI DD DSN=your.m4repi,DISP=(OLD,DELETE)
14 //MAINPUT DD *
CONTROL
FILE REPORT
/*
/* ADDITIONAL REPORT MANAGER JCL STATEMENTS
/*
15 //M4PRINT DD SYSOUT=*          COLLATE ONLY OUTPUT/DEFAULT ROUTE OUTPUT
16 //DEST001 DD SYSOUT=*          ROUTE DESTINATION STATEMENTS AS NEEDED

```

Figure 2-25 Report Manager Three-Step JCL with Collating, with or without Routing

An explanation of the numbered statements for [Figure 2-25](#) follows:

Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors, and generates the necessary sort control statements. The execution of VISION:Builder is controlled by run control statements that specify the files and DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements, diagnostic messages and the report(s).
4	M4OLD defines the old master file and is the primary data input file to the processing step in this run.
5	The M4SORT file contains the sort control statements and COLLATE commands for the COLLATOR (SORT) program.
6	M4REPO is the file containing the VISION:Builder report file output from the processing step. This includes the report command language and ROUTE commands necessary to build and route the reports from the processing step. The information and data are used in subsequent sort and report generation steps.
7	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements. As part of the run control statements for the application, the report manager COLLATE (and ROUTE) commands are specified.
8	This statement calls in and executes the report manager COLLATOR program which invokes the system sort program.
9	The SYSIN file contains the sort control statements and the COLLATE commands for the COLLATOR (SORT) program.
10	The M4LIST statement defines the location of the system output device for the COLLATOR program to direct any messages and statistics.
11	VISION:Builder is invoked to generate the report.
12	This M4LIST is output from the report step and includes a listing of the run control statements and report messages.
13	Defines the location of the sorted report file (M4REPI), output by the sort step, that is the input to the report step.

Notes	Explanation
14	<p>This M4INPUT indicates to VISION:Builder that only the report step has to be executed.</p> <p>Note: SORTOUT should not have DCB information on the JCL.</p>
15	<p>The M4PRINT statement defines a location for Report Manager output.</p> <ul style="list-style-type: none"> ■ In collate-only runs, the collated reports are directed here first followed by the non-collated reports. ■ In runs that perform routing, the non-routed reports are directed here. ■ If the M4PRINT statement is present without any report manager commands in the application source code, all reports are directed here instead of M4LIST. <p>Note: The report output from VISION:Builder can be directed to SYSOUT, disk and tape data sets, PDS members, or wherever your operating system JCL specification will allow.</p>
16	<p>The DEST001 statement defines the destination for report REP3.</p> <p>Additional statements are needed for each designated destination name specified in the report manager ROUTE commands.</p>

Report Manager Three-Step JCL for Routing Only

In three-step application runs, minor JCL changes are needed for the Report Manager when performing routing only.

There are no changes needed in the processing step; however, your application source code includes your collating and routing specifications.

There are no changes needed in the sort step.

Finally, the report step needs JCL statements that are the destinations for the Routed reports. The M4PRINT DD statement receives any non-routed reports. The specifically named DD statements receive routed reports that correspond to the designated routing destinations.

Figure 2-26 shows a simple three-step JCL example with the additional JCL changes needed for the report manager.

Notes

```

//      JOB (accounting information)
//*
//JOBLIB DD DSN=your.builder.loadlib,DISP=SHR
//*
//* JCL for THREE-STEP APPLICATION RUN with REPORT MANAGER
//*
//* PROCESSING STEP - ROUTE Commands present
//*      (no COLLATE Commands are coded)
//*
1 //stepa EXEC PGM=MARKIV,REGION=1536K
2 //M4LIB DD DSN=your.m4lib,DISP=SHR
3 //M4LIST DD SYSOUT=*
4 //M4OLD DD DSN=old.master.file,DISP=(OLD,KEEP)
5 //M4SORT DD DSN=sort.file,DISP=(NEW,PASS),UNIT=sysda,
//      SPACE=(TRK,n)
6 //M4REPO DD DSN=your.m4repo,DISP=(NEW,PASS),UNIT=sysda,
//      SPACE=(TRK,(n,n))
7 //M4INPUT DD *
CONTROL SORT EXTERNAL
FILE MASTER INFUT, NAME . . .
FILE REPORT
ROUTE REPORT REP1 TO DEST001
;
REP1: REPORT . . .
. . .
END REPORT
;
REP2: REPORT . . .
. . .
END REPORT
;
REP3: REPORT . . .
. . .
END REPORT
/*
//*
//* SORT STEP
//*
8 //stepb EXEC PGM=COLLATOR,REGION=1536K
//*
//SYSIN DD DSN=sort.file,DISP=(,PASS)
//SORTIN DD DSN=your.m4repo,DISP=(OLD,DELETE)
//SORTOUT DD DSN=your.m4repl,DISP=(NEW,PASS),
//      UNIT=sysda,SPACE=(TRK,(n,n))
//SYSOUT DD SYSOUT=*
//SORTLIB DD DSN=SYS1,SORTLIB,DISP=SHR
//SORIWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORIWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORIWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//*
//* REPORT STEP
//*
9 //stepc EXEC PGM=MARKIV,REGION=1536K
10 //M4LIST DD SYSOUT=*
11 //M4REPI DD DSN=your.m4repi,DISP=(OLD,DELETE)
12 //M4INPUT DD *
CONTROL
FILE REPORT
/*
//* ADDITIONAL REPORT MANAGER JCL STATEMENTS
//*
13 //M4PRINT DD SYSOUT=*      DEFAULT ROUTE OUTPUT
14 //DEST001 DD SYSOUT=*      ROUTE DESTINATION STATEMENTS AS NEEDED

```

Figure 2-26 Report Manager Three-Step JCL with Routing Only

An explanation of the numbered statements for [Figure 2-26](#) follows:

Notes	Explanation
1	VISION:Builder processes the source statements from the M4INPUT data set, checking for syntactical errors, and generates the necessary sort control statements. The execution of VISION:Builder is controlled by run control statements that specify the files and DD statements required for this job step.
2	The common library defines the cataloged data set on a direct access device. This is the same data set name as required for the file definition.
3	The M4LIST statement defines the location of the system output device for this job and includes a listing of the source statements, diagnostic messages and the report(s).
4	M4OLD defines the old master file and is the primary data input file to the processing step in this run.
5	The M4SORT file contains the sort control statements for the sort program.
6	M4REPO is the file containing the VISION:Builder report file output from the processing step. This includes the report command language and ROUTE commands necessary to build and route the reports from the processing step. The information and data are used in subsequent sort and report generation steps.
7	The M4INPUT data set consists of VISION:Builder source statements. The CONTROL statement must be the first of these source statements. As part of the run control statements for the application, the report manager ROUTE commands are specified.
8	This statement calls in and executes the system sort program.
9	VISION:Builder is invoked to generate the report.
10	This M4LIST is output from the report step and includes a listing of the run control statements and report messages.
11	Defines the location of the sorted report file (M4REPI), which is output by the sort step and the input to the report step.

Notes	Explanation
12	This M4INPUT indicates to VISION:Builder that only the report step has to be executed.
13	<p>The M4PRINT statement defines a location for Report Manager output.</p> <ul style="list-style-type: none"> ■ In runs that perform routing, the non-routed reports are directed here. ■ If the M4PRINT statement is present without any report manager commands in the application source code, all reports are directed here instead of M4LIST. <p>Note: The report output from VISION:Builder can be directed to SYSOUT, disk and tape data sets, PDS members, or wherever your operating system JCL specification will allow.</p>
14	<p>The DEST001 statement defines the destination for report REP1.</p> <p>Additional statements are needed for each designated destination name specified in the report manager ROUTE commands.</p>

Alternate Report Output Method JCL Examples

Alternate Report Output Methods Single-Step JCL

In single-step application runs using one or more alternate report output methods, you must include JCL statements that define the destination for each alternate output. If HTML is used as the method for any alternate output, you must add a JCL statement that identifies the location of the HTML templates used to prepare the HTML output. A message identifying the destination for each alternate output and any messages generated by the alternate output method are directed to M4LIST. Note that a single VISION:Builder application program may generate multiple reports using alternate report output methods in any combination along with print-formatted reports, up to a total of 255 such outputs.

[Figure 2-27](#) shows a single-step JCL example with the JCL needed for the different alternate output methods:

Notes

```

//jobname JOB ...
//JOBLIB DD DSN=your.builder.loadlib,DISP=SHR
/**
/** JCL for SINGLE-STEP PROCESSING RUN using ALTERNATE
/** REPORT OUTPUT METHODS (Process-Sort-Report)
/**
1 //step EXEC PGM=MARKIV,REGION=2M
2 //M4LIB DD DSN=your.m4lib,DISP=SHR
3 //M4LIST DD SYSOUT=*
4 //M4OLD DD DSN=old.master.file,DISP=SHR
//SYSOUT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
5 //M4REPO DD UNIT=SYSDA,SPACE=(CYL,(n,n))
6 //M4INPUT DD *
CONTROL
FILE MASTER INPUT, NAME . . .
FILE REPORT
;
REPORT . . .
FORMAT METHOD . . ., DDNAME . . .
END REPORT
.
.
/**
/** ADDITIONAL JCL STATEMENTS FOR COMMA-DELIMITED OUTPUT
/**
7 //comdata DD DSN=comma.delimit.data,DISP=(NEW,CATLG),
// UNIT=SYSDA,SPACE=(CYL,(n,n))
/**
/** ADDITIONAL JCL STATEMENTS FOR TAB-DELIMITED OUTPUT
/**
8 //tabdata DD DSN=tab.delimit.data,DISP=(NEW,CATLG),
// UNIT=SYSDA,SPACE=(CYL,(n,n))
/**
/** ADDITIONAL JCL STATEMENTS FOR HTML OUTPUT
/**
9 //htmlout DD DSN=html.document.output,DISP=(NEW,CATLG),
// UNIT=SYSDA,SPACE=(CYL,(n,n,m))
10 //M4HTBASE DD DSN=your.html.template.library,DISP=SHR
/**
/** ADDITIONAL JCL STATEMENTS FOR PLAIN TEXT OUTPUT
/**
11 //plaintxt DD DSN=plain.text.output,DISP=(NEW,CATLG),
// UNIT=SYSDA,SPACE=(CYL,(n,n))
/**
/** ADDITIONAL JCL STATEMENTS FOR RAW DATA OUTPUT
/**
12 //rawout DD DSN=raw.data.output,DISP=(NEW,CATLG),
// UNIT=SYSDA,SPACE=(CYL,(n,n))
/**
/** OPTIONAL JCL STATEMENTS FOR HTML OUTPUT
/**
13 //M4WORK1 DD UNIT=SYSDA,SPACE=(CYL,(n,n)) HTML
14 //M4OPTS DD DSN=your.builder.worklib(M4OPTNS),DISP=SHR
//

```

Figure 2-27 Alternate Report Output Methods Single-Step Application Run JCL

An explanation of the numbered statements in [Figure 2-27](#) follows.

Notes	Explanation
1	This statements invokes VISION:Builder to process the source statements from the M4INPUT data set, check for syntactical errors, process the data base input records, generate the necessary sort control parameters, extract and sort the required report data, and generate any report output. The execution of VISION:Builder is controlled by run control statements that specify the files and DD statements required for this job step.
2	The M4LIB statement identifies the cataloged data set on a direct access device containing the common library objects needed by this application.
3	The M4LIST statement defines the location of the system output device for this job that receives the listing of the source statements, diagnostic messages, and the run statistics.
4	The M4OLD statement identifies the master file and is the primary data input to the processing step in the run. If the primary data input is through a database manager, JCL statements required by the database manager are used in place of this statement.
5	The M4REPO statement defines the file containing the report command language. The report data records are passed directly to the sort process.
6	The M4INPUT statement identifies the source of the VISION:Builder source statements. The CONTROL statement must be the first of these source statements. A single-step run requires that the default SORT INTERNAL be allowed. Other run control statements and any valid requests typically follow the CONTROL statement.
7	This JCL statement is an example of one that would be required for comma-delimited output (CSV). The ddname must be identical to the name specified with the DDNAME keyword on the FORMAT METHOD CSV statement.
8	This JCL statement is an example of one that would be required for tab-delimited output. The ddname must be identical to the name specified with the DDNAME keyword on the FORMAT METHOD TAB statement.
9	This JCL statement is an example of one that would be required for HTML output. Note that this statement must define a partitioned data set and the ddname must be identical to the name specified with the DDNAME keyword on the FORMAT METHOD HTML statement. VISION:Builder creates up to 4 members in the data set. See Chapter 13, VISION:Builder HTML Document Style Customization for a discussion of the content of each created member.

Notes	Explanation
10	<p>The M4HTBASE statement identifies the data set containing the HTML template members required for the styles used by this application. This data set must be a partitioned data set containing the required members. See Chapter 13, VISION:Builder HTML Document Style Customization for a discussion of the names of the members required in this data set, their function, and content.</p>
11	<p>This JCL statement is an example of one that would be required for plain text output. The ddname must be identical to the name specified with the DDNAME keyword on the FORMAT METHOD PLAINTEXT statement.</p>
12	<p>This JCL statement is an example of one that would be required for Raw Data output. The ddname must be identical to the name specified with the DDNAME keyword on the FORMAT METHOD RAWDATA statement.</p>
13	<p>The M4WORK1 statement defines a work data set used by the HTML output method. If the statement is omitted, VISION:Builder dynamically allocates the data set as necessary, with a space allocation of (CYL,(1,1)) and a unit name of SYSDA.</p>
14	<p>The M4OPTS statement identifies an optional input stream containing VISION:Builder run time parameters and/or options. If this statement is omitted, the default parameters and options are used. The acceptable parameters are:</p> <p>UNITNAME Defines the DASD Pool Name for dynamic dataset allocation. The default name is SYSDA.</p> <p>URLFILE Defines the type of URL to create for HTML document file references. Acceptable values are HFS or MVS. Use MVS or HFS if the document will be served up by WebSphere running under z/OS or OS/390. Use HFS if the document will be served up by a web server running on another platform.</p> <p>FILESIZE Defines the file size limit for the files or members containing the body frame of an HTML document. Data for large reports will be segmented into files or members not exceeding the specified size. The default size is 500K.</p> <p>See the sample member named M4OPTNS in the installation WORKLIB dataset for further details.</p>

Alternate Report Output Methods Three-Step JCL

In three-step application runs using one or more alternate report output methods, no changes are needed in the processing and sort step JCL. In the report step, you must include JCL statements that define the destination for each alternate output. If HTML is used as the method for any alternate output, you must add a JCL statement that identifies the location of the HTML templates used to prepare the HTML output. A message identifying the destination for each alternate output and any messages generated by the alternate output method are directed to M4LIST. Note that a single VISION:Builder application program may generate multiple reports using alternate report output methods in any combination along with print-formatted reports, up to a total of 255 such outputs.

[Figure 2-28](#) The following single-step JCL example shows the JCL needed for the different alternate output methods:

Notes

```

//jobname JOB ...
//JOBLIB DD DSN=your.builder.loadlib,DISP=SHR
//*
//* JCL for THREE-STEP PROCESSING RUN using ALTERNATE
//* REPORT OUTPUT METHODS
//*
//* PROCESS STEP
//*
1 //stepa EXEC PGM=MARKIV,REGION=2M
2 //M4LIB DD DSN=your.m4lib,DISP=SHR
3 //M4LIST DD SYSOUT=*
4 //M4OLD DD DSN=old.master.file,DISP=SHR
5 //M4REPO DD UNIT=SYSDA,SPACE=(CYL,(n,n)),DISP=(NEW,PASS)
6 //M4SORT DD UNIT=SYSDA,SPACE=(TRK,(1,1)),DISP=(NEW,PASS)
7 //M4INPUT DD *
CONTROL SORT EXTERNAL
FILE MASTER INPUT, NAME . . .
FILE REPORT
;
REPORT . . .
FORMAT METHOD . . ., DDNAME . . .
END REPORT
//*
//* SORT STEP
//*
8 //stepb EXEC PGM=SORT,REGION=4M
9 //SYSIN DD DSN=*.stepa.M4SORT,DISP=(OLD,DELETE)
10 //SORTIN DD DSN=*.stepa.M4REPO,DISP=(OLD,DELETE)
11 //SORTOUT DD UNIT=SYSDA,SPACE=(CYL,(n,n)),DISP=(NEW,PASS)
//SYSOUT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,n,,CONTIG)
//SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
//*
//* REPORT STEP
//*
12 //stepc EXEC PGM=MARKIV,REGION=1M
13 //M4LIST DD SYSOUT=*
14 //M4REPI DD DSN=*.stepb.SORTOUT,DISP=(OLD,DELETE)
15 //M4INPUT DD *
CONTROL
FILE REPORT
//*
//* ADDITIONAL JCL STATEMENTS FOR COMMA-DELIMITED OUTPUT
//*
16 //comdata DD DSN=comma.delimit.data,DISP=(NEW,CATLG),
// UNIT=SYSDA,SPACE=(CYL,(n,n))
//*
//* ADDITIONAL JCL STATEMENTS FOR TAB-DELIMITED OUTPUT
//*
17 //tabdata DD DSN=tab.delimit.data,DISP=(NEW,CATLG),
// UNIT=SYSDA,SPACE=(CYL,(n,n))
//*
//* ADDITIONAL JCL STATEMENTS FOR HTML OUTPUT
//*
18 //htmlout DD DSN=html.document.output,DISP=(NEW,CATLG),
// UNIT=SYSDA,SPACE=(CYL,(n,n,m))
19 //M4HTBASE DD DSN=your.html.template.library,DISP=SHR
//*
//* ADDITIONAL JCL STATEMENTS FOR PLAIN TEXT OUTPUT
//*
20 //plaintxt DD DSN=plain.text.output,DISP=(NEW,CATLG),
// UNIT=SYSDA,SPACE=(CYL,(n,n))
//*
//* ADDITIONAL JCL STATEMENTS FOR RAW DATA OUTPUT

```

```

21      /*
      //raw      DD DSN=raw.data.output,DISP=(NEW,CATLG),
      //          UNIT=SYSDA,SPACE=(CYL,(n,n))
      /*
      /* OPTIONAL JCL STATEMENTS FOR HTML OUTPUT
      /*
22      //M4WORK1 DD UNIT=SYSDA,SPACE=(CYL,(n,n)) HTML
23      //M4OPTS  DD DSN=your.builder.worklib(M4OPTNS),DISP=SHR
      //
  
```

Figure 2-28 Alternate Report Output Methods Three-Step Application Run JCL

An explanation of the numbered statements in [Figure 2-28](#) follows.

Notes	Explanation
1	This statement invokes VISION:Builder to process the source statements from the M4INPUT data set, check for syntactical errors, process the database input records, extract the report data, and generate the necessary sort control parameters. The execution of VISION:Builder is controlled by run control statements that specify the files and DD statements required for this job step.
2	The M4LIB statement identifies the cataloged data set on a direct access device containing the common library objects needed by this application.
3	The M4LIST statement defines the destination for the output stream of this job that receives the listing of the source statements, diagnostic messages, and run statistics.
4	The M4OLD statement identifies the master file and is the primary data input to the processing step in the run. If the primary data input is through a database manager, JCL statements required by the database manager are used in place of this statement.
5	The M4REPO statement defines the file containing the report command language and the extracted data records. This file is used as input to the sort program in the following step.
6	The M4SORT statement defines the file containing the sort control statements used by the sort program in the following step.
7	The M4INPUT statement identifies the source of the VISION:Builder source statements. The CONTROL statement must be the first of these source statements. For a three-step run, you must specify SORT EXTERNAL on the CONTROL statement. Other run control statements and valid requests typically follow the CONTROL statement.
8	This statement invokes the system sort program.
9	The SYSIN statement for the sort program identifies the data set containing the sort control statements created by VISION:Builder in the previous step.

Notes	Explanation
10	The SORTIN statement identifies the input file containing the records to be sorted. This file was created by VISION:Builder in the previous step.
11	The SORTOUT statement defines the output data set containing the command language and extracted data records in sorted order. This file is used in the subsequent report generation step.
12	This statement invokes the report generation process of VISION:Builder.
13	The M4LIST statement defines the destination for the output stream of this job that receives the run control statement listing, diagnostic messages, and print-formatted reports not directed to other destinations.
14	The M4REPI statement identifies the data set containing the sorted command language records and extracted data records that will be used to create the various report outputs.
15	The M4INPUT statement identifies the source of the run control statement that directs VISION:Builder to perform its report generation function.
16	This JCL statement is an example of one that is required for comma-delimited output (CSV). The ddname must be identical to the name specified with the DDNAME keyword on the FORMAT METHOD CSV statement.
17	This JCL statement is an example of one that is required for tab-delimited output. The ddname must be identical to the name specified with the DDNAME keyword on the FORMAT METHOD TAB statement.
18	This JCL statement is an example of one that would be required for HTML output. Note that this statement must define a partitioned data set and the ddname must be identical to the name specified with the DDNAME keyword on the FORMAT statement. VISION:Builder creates up to 4 members in this data set. See Chapter 13, VISION:Builder HTML Document Style Customization for a discussion of the content of each created member.
19	The M4HTBASE statement identifies the data set containing the HTML template members required for the styles used by this application. This data set must be a partitioned data set containing the required members. See Chapter 13, VISION:Builder HTML Document Style Customization for a discussion of the names of the members required in this data set, their function, and content.

Notes	Explanation
20	This JCL statement is an example of one that would be required for plain text output. The ddname must be identical to the name specified with the DDNAME keyword on the FORMAT METHOD PLAINTEXT statement.
21	This JCL statement is an example of one that would be required for Raw Data output. The ddname must be identical to the name specified with the DDNAME keyword on the FORMAT METHOD RAWDATA statement.
22	The M4WORK1 statement defines a work data set used by the HTML output method. If the statement is omitted, VISION:Builder dynamically allocates the data set as necessary, with a space allocation of (CYL,(1,1)) and a unit name of SYSDA.
23	The M4OPTS statement identifies an optional input stream containing VISION:Builder run time parameters and/or options. If this statement is omitted, the default parameters and options are used. The acceptable parameters are:
UNITNAME	Defines the DASD Pool Name for dynamic dataset allocation. The default name is SYSDA.
URLFILE	Defines the type of URL to create for HTML document file references. Acceptable values are HFS or MVS. Use MVS or HFS if the document will be served up by WebSphere running under z/OS or OS/390. Use HFS if the document will be served up by a web server running on another platform.
FILESIZE	Defines the file size limit for the files or members containing the body frame of an HTML document. Data for large reports will be segmented into files or members not exceeding the specified size. The default size is 500K.
	See the sample member named M4OPTNS in the installation WORKLIB dataset for further details.

VISION:Builder – IMS Database Interface and Retrieval

Note: Examples in this chapter will be illustrated using VISION:Workbench™ for DOS. VISION:Workbench is a VISION:Builder application development system executing on IBM PCs and compatibles. For more information about VISION:Workbench, see the [VISION:Workbench for DOS Reference Guide](#).

MARKDLI, DBDs, and PSBs

A VISION:Builder interface routine, MARKDLI, is supplied as part of the Data Base Interface/IMS and Data Base Retrieval/IMS option. This routine is link edited with the IMS interface routine (DFSLI000) to form the interface between VISION:Builder and IMS. VISION:Builder operates as a normal batch program under the IMS region controller and uses your installation's IMS system to access DL/I files.

The VISION:Builder/IMS operations process is shown in [Figure 3-1](#).

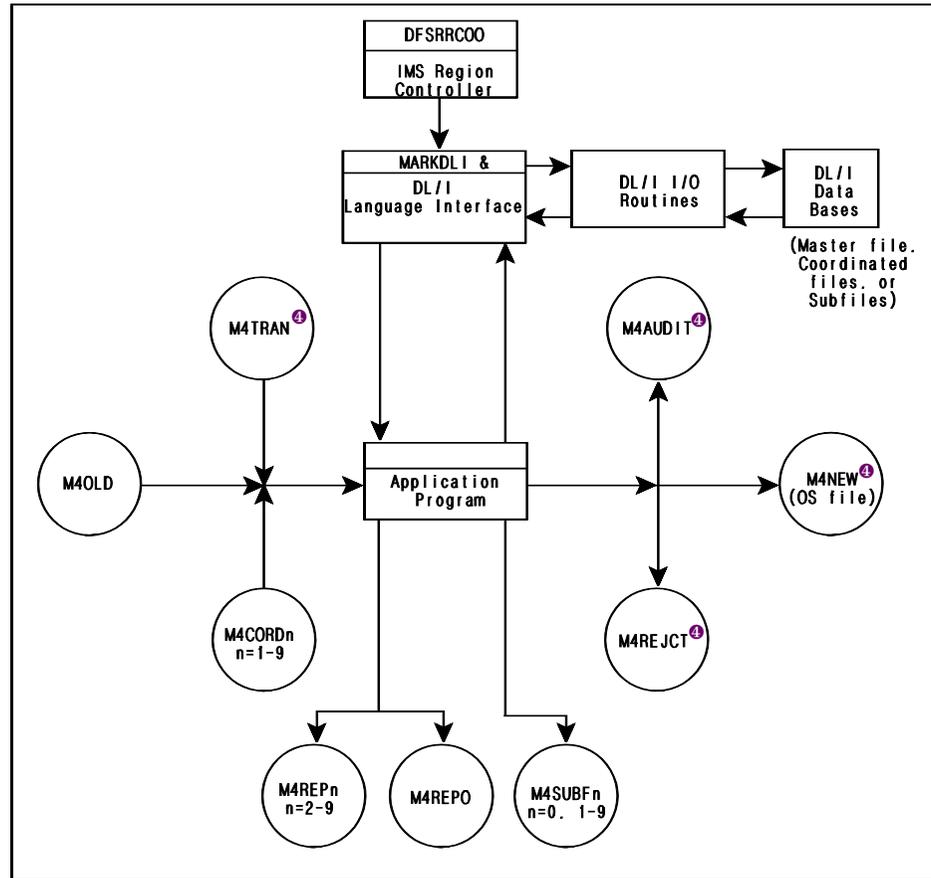


Figure 3-1 VISION:Builder - IMS Operation

The database definitions you provide to IMS and VISION:Builder must be compatible with each other. This means that the DBD and PSB provided to IMS must be compatible with the VISION:Builder file definition. [Figure 3-2](#) illustrates the relationship between an IMS DBD, PSB, and a VISION:Builder file definition.

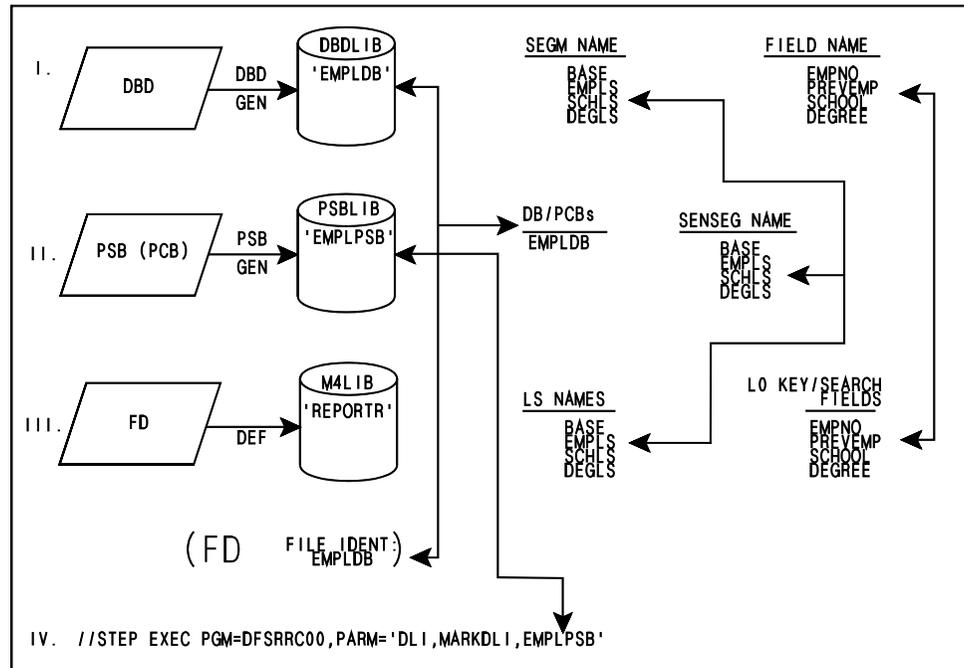


Figure 3-2 VISION:Builder and IMS

Sample File Definition

Figure 3-3 shows the file definition for a personnel file that has a three level hierarchical structure. The DL/I organization method chosen is HISAM. Since the IMS definition process calculates an optimum record size and blocking factor, these entries have not been provided. Note that each segment has only one key field and that segments PRIOREMP and MINOR do not have count fields associated with them. The DBD name (EMPLYDBD in this example) is required in the file identification specification.

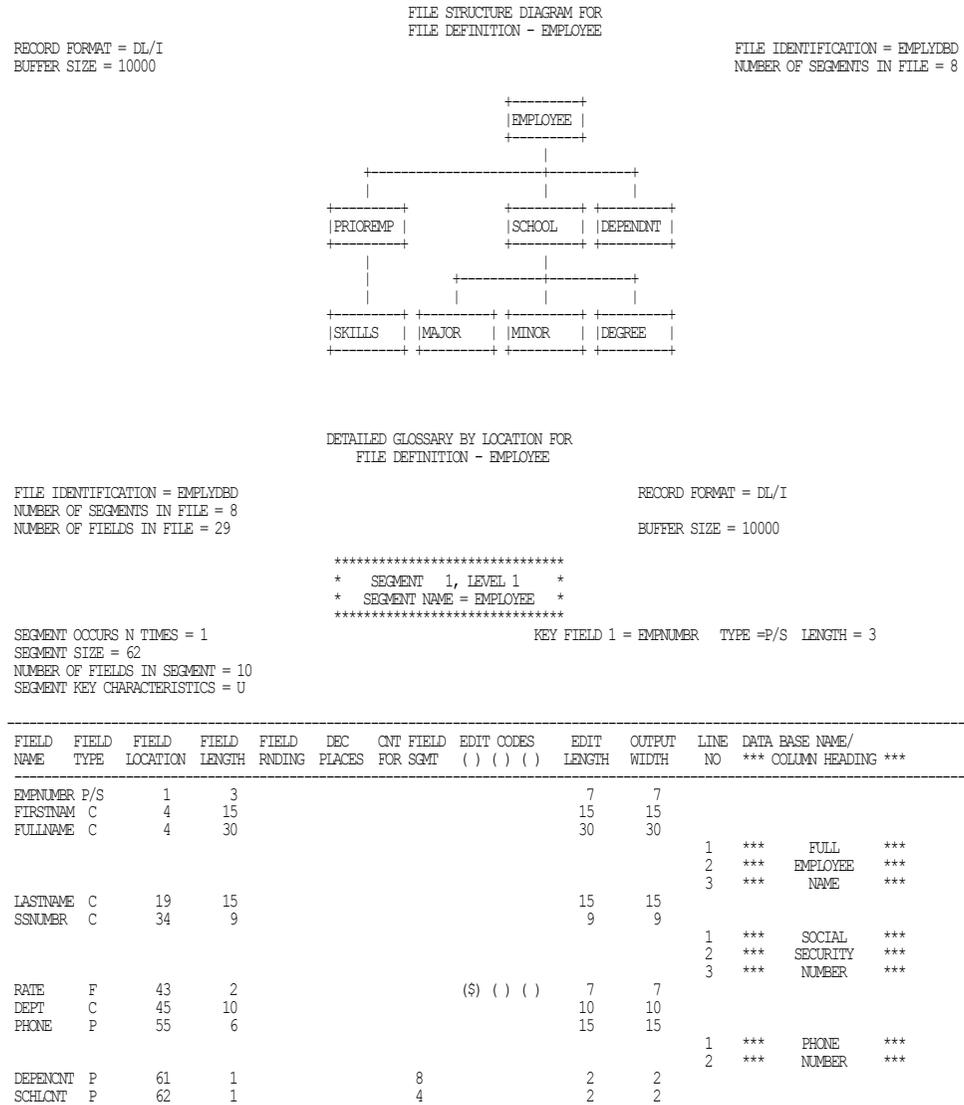


Figure 3-3 File Definition for Sample Application (Page 1 of 3)

DETAILED GLOSSARY BY LOCATION FOR
FILE DEFINITION - EMPLOYEE

* SEGMENT 2, LEVEL 2 *
* SEGMENT NAME = PRIOREMP *

COUNT FIELD FOR SEGMENT = **NONE** KEY FIELD 1 = PREVEMP TYPE =C/S LENGTH = 20
SEGMENT SIZE = 66
NUMBER OF FIELDS IN SEGMENT = 5
SEGMENT KEY CHARACTERISTICS = U

FIELD NAME	FIELD TYPE	FIELD LOCATION	FIELD LENGTH	FIELD RNDING	DEC PLACES	CNT FOR	FIELD FOR SGMT	EDIT CODES () () ()	EDIT LENGTH	OUTPUT WIDTH	LINE NO	DATA BASE NAME/ *** COLUMN HEADING ***
PREVEMP	C/S	1	20						20	20	1	*** PREVIOUS EMPLOYER ***
ADDRESS1	C	21	20						20	20	2	***
ADDRESS2	C	41	20						20	20		***
ZIP	C	61	5						5	5		***
SKILLCNT	P	66	1			3			2	2		***

DETAILED GLOSSARY BY LOCATION FOR
FILE DEFINITION - EMPLOYEE

* SEGMENT 3, LEVEL 3 *
* SEGMENT NAME = SKILLS *

COUNT FIELD FOR SEGMENT = SKILLCNT KEY FIELD 1 = SKILL TYPE =C/S LENGTH = 20
SEGMENT SIZE = 22
NUMBER OF FIELDS IN SEGMENT = 2
SEGMENT KEY CHARACTERISTICS = U

FIELD NAME	FIELD TYPE	FIELD LOCATION	FIELD LENGTH	FIELD RNDING	DEC PLACES	CNT FOR	FIELD FOR SGMT	EDIT CODES () () ()	EDIT LENGTH	OUTPUT WIDTH	LINE NO	DATA BASE NAME/ *** COLUMN HEADING ***
SKILL	C/S	1	20						20	20	1	*** SKILL ***
RATE/HR	P	21	2		2				5	5		***

DETAILED GLOSSARY BY LOCATION FOR
FILE DEFINITION - EMPLOYEE

* SEGMENT 4, LEVEL 2 *
* SEGMENT NAME = SCHOOL *

COUNT FIELD FOR SEGMENT = SCHLCNT KEY FIELD 1 = SCHOOL TYPE =C/S LENGTH = 20
SEGMENT SIZE = 23
NUMBER OF FIELDS IN SEGMENT = 4
SEGMENT KEY CHARACTERISTICS = U

FIELD NAME	FIELD TYPE	FIELD LOCATION	FIELD LENGTH	FIELD RNDING	DEC PLACES	CNT FOR	FIELD FOR SGMT	EDIT CODES () () ()	EDIT LENGTH	OUTPUT WIDTH	LINE NO	DATA BASE NAME/ *** COLUMN HEADING ***
SCHOOL	C/S	1	20						20	20	1	*** SCHOOL ***
MAJCOUNT	P	21	1			5			2	2		***
MINCOUNT	P	22	1			6			2	2		***
DEGCOUNT	P	23	1			7			2	2		***

Figure 3-3 File Definition for Sample Application (Page 2 of 3)

In processing an IMS database, IMS returns status codes to VISION:Builder in the PCB. If an unexpected status code occurs during transaction application with MOSAIC processing, the transaction is rejected, but VISION:Builder can continue.

Unexpected status codes occurring at other times cause VISION:Builder to terminate the job with a user abend code of 0111. Prior to termination, VISION:Builder issues an error message that includes the unexpected status code. Usually, the status code indicates a problem with JCL or with the database structures.

If the information in the message is not sufficient for debugging, a dump should be requested. In this dump, register 7 contains the address of the DL/I call parameter list.

IMS Batch Region Execution of VISION:Builder

Since VISION:Builder operates under the IMS region controller, the program DFSRRC00 is invoked rather than VISION:Builder. The DD statements for VISION:Builder data sets are the same as required in a normal VISION:Builder job, except for those files specified as IMS files.

These statements are replaced by the DD statements as specified in the DBD for the file. Additional DD statements are required to specify the PSB and DBD libraries. The ddname used is IMS. The JCL for a processing run is shown in [Figure 3-4](#). The ddnames for the DL/I databases differ depending on the DBD. The EXEC statement must be filled out as follows:

```
//mk4 EXEC PGM=DFSRRC00,PARM='DLI,MARKDLI,psbname'
```

where psbname is the name of the PSB previously link edited into the PSB library.

In all cases, the JCL is the same as shown for the batch region in [Figure 3-4](#).

Notes

```

//markdli JOB (accounting information)
//JOBLIB DD DSN=IMSVS.RESLIB,DISP=SHR
// DD DSN=your.builder.loadlib,DISP=SHR
1 //mk4 EXEC PGM=DFSRRRC00,PARM='DLI,MARKDLI,psbname',
// REGION=1536K
//M4LIST DD SYSOUT=a
//M4LIB DD DSN=your.m4lib,DISP=SHR
//M4REPO DD UNIT=sysda,SPACE=(TRK,(nn,nn)),
// DISP=(NEW,PASS),
// DSN=your.m4repo
//M4SORT DD UNIT=sysda,SPACE=(TRK,(nn,nn)),
// DISP=(NEW,PASS),
// DSN=your.m4sort
2 //IMS DD DSN=ims.psbllib,DISP=SHR
// DD DSN=ims.dbdlib,DISP=SHR
//DFSRESLB DD DSN=IMSVS.RESLIB,DISP=SHR
//IEFRDER DD DUMMY
//DBASE1 DD DSN=data.base1,disp=SHR
//DBASE2 DD DSN=data.base2,disp=SHR
//M4INPUT DD *
CONTROL
FILE MASTER INPUT, NAME EMPLOYEE
FILE REPORT
.
.
/*
//

```

Figure 3-4 JCL for an IMS Processing Run (Batch Region)

The following explanations are keyed to the numbered statements in [Figure 3-4](#).

Notes	Explanation
1	DFSRRRC00 is the IMS executable program. Parm DLI is used when application control blocks are built dynamically in execution. MARKDLI is the VISION:Builder entry module that provides linkage between IMS and VISION:Builder. Psbname is your link edited PSB.
2	IMS is the ddname for the data sets containing the link edited DBDs and PSBs.

Note: If the application control blocks have been prebuilt (that is, an ACBGEN has been executed for the DBDs and PSBs), change parm DLI to DBB and change the IMS DD statement to contain the name of the output data set from the ACBGEN (on the IMSACB DD statement).

IMS BMP Region Execution of VISION:Builder

To execute VISION:Builder in a batch message processing (BMP) region, you must run an IMSGEN update. This update must include the DBD name for the database macro and the PSB name for the application macro (BATCH must be used as the program type). Also, you must supply all DD and data set names for IMS user files. In a BMP region, VISION:Builder has access to those online files and to the online message queues (through the IMS control region), as well as access to any offline standard OS files in the application JCL.

For example, database and application macros could be as follows:

```
DBDNAME=EMPLYDBD

APPLCTN PSB=psbname
        PGMTYPE=BATCH
```

IMS user file DD statements are added to the IMS/ESA dynamic JCL allocation pool, for example:

```
//DLIDD1 DD DSN=EMPLOYEE.PRIME,DISP=SHR
//DLIDD1 DD DSN=EMPLOYEE.OVERFLOW,DISP=SHR
```

VISION:Builder job control does not require any DD statements for online IMS user files (this is handled by the IMS region controller).

You must assign a sufficient region size to properly execute in the BMP. Refer to the appropriate IMS manual for a description of each of the parameters on the PARM statement.

```
//markbmp JOB (accounting information)
//JOBLIB DD DSN=IMSVS.RESLIB,DISP=SHR
// DD DSN=your.builder.loadlib,DISP=SHR
//MK4 EXEC PGM=DFSRRC00,REGION=1536K,
        PARM='BMP,MARKDLI,psbname,,,N00000'
//IEFRDER DD DUMMY
//DFSRESLB DD DSN=IMSVS.RESLIB,DISP=SHR
//IMS DD DSN=IMSVS.ACBLIB,DISP=SHR
//M4LIST DD DSN=SYSOUT=*
//M4LIB DD DSN=your.m4lib,DISP=SHR
//M4REPO DD DSN=your.m4repo,DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(TRK,(nn,nn),RLSE)
//M4SORT DD DSN=your.m4sort,DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(TRK,(nn,nn),RLSE)
//M4INPUT DD *
CONTROL SORT EXTERNAL
FILE MASTER INPUT, NAME EMPLOYEE
FILE REPORT
.
.
.
/*
//
```

Figure 3-5 JCL for an IMS BMP Processing Run

VISION:Builder Extended DL/I Support

VISION:Builder extended DL/I support expands capabilities for processing IMS databases, including virtual keys used with secondary indexing, alias names used with non-unique IMS key/search fields, and generic and duplicate root keys for file retrieval. The following subsections describe these functions.

Secondary Indexing

VISION:Builder Data Base Interface and Data Base Retrieval supports secondary indices, with no modifications, as long as the source and target segments are the same and the search field (XDFLD) is composed of one or more contiguous fields.

If you define a virtual key to VISION:Builder, processing is made available for secondary indices. When using secondary indices, the entry point (target segment) can be one segment and the key fields can come from a dependent segment (source segment), or the source key could have noncontiguous fields.

A virtual key is defined as a type V segment key on the L0 statement in a VISION:Builder file definition (see the [VISION:Builder Reference Summary](#)). A virtual key defined to VISION:Builder must also be an XDFLD search field in the secondary index DBD. VISION:Builder treats the virtual key as any other database segment key.

During IMS calls, the virtual key is not included in the segment and does not appear in the IMS file on disk, nor can it be overdefined in the VISION:Builder file definition.

Non-Unique IMS Key/Search Field Names

VISION:Builder requires the field names in a file definition to be unique. VISION:Builder allows you to specify alias names for key/search field names when non-unique key and search field names exist in an IMS database. The non-unique name or names must be unique within each segment and are valid only for key and search fields. The aliases are used for segment search arguments (SSA) and cannot be referenced in your VISION:Builder application; the actual field name must be used. An alias is defined on an L0 statement in your VISION:Builder file definition.

Generic and Duplicate Root Keys

A generic key is the high order part of a record key whose length is less than the full key. Generic keys can only be specified for the root segment. They are defined in your VISION:Builder file definition as an overdefinition of the primary key.

Random access of an IMS file using a generic key search is accomplished by the use of an additional file (RF) statement, with the generic key specified as the key/search field. Read operators (described in the [VISION:Builder Reference Guide](#)) are used in your VISION:Builder program to search for the generic key.

The same read operators used for generic key support can also be used for direct retrieval of all occurrences of duplicate root segment keys (refer to the [*VISION:Builder Reference Guide*](#)).

Note: Examples in this chapter will be illustrated using VISION:Workbench for DOS. VISION:Workbench for DOS is a VISION:Builder application development system executing on IBM PCs and compatibles. For more information about VISION:Workbench for DOS, see the [VISION:Workbench for DOS Reference Guide](#).

ESDS and KSDS Files

VISION:Builder supports the use of VSAM records for key sequenced data sets (KSDS) and entry sequenced data sets (ESDS). This includes spanned records. A record format entry of K for KSDS or E for ESDS is entered on the VISION:Builder FD statement. The VSAM cluster must have been defined properly before VISION:Builder can access the records. Refer to the *IBM Access Methods Services* manual for information on the definition of VSAM clusters.

ESDS and KSDS Alternate Index Paths

VISION:Builder also supports alternate index paths to access VSAM KSDS or ESDS. VISION:Builder treats an ESDS alternate index path as though it is a KSDS. The ESDS alternate path must be defined to VISION:Builder as a KSDS.

[Figure 4-1](#) illustrates a sample alternate index definition for the VISION:Builder file definition in [Figure 4-2](#).

```
//STEP1 EXEC PGM=IDCAMS,REGION=1536K
DEFINE ALTERNATEINDEX -
  (NAME (vsam.aix) -
  RELATE (base.cluster) -
  KEYS (3 24) -
  RECORDSIZE (80 80) -
  VOL (volser) -
  TRACKS (n n) -
  NONUNIQUEKEY -
  UPGRADE) -
DATA -
  (NAME (vsam.aix.data)) -
CATALOG (catname)
/*
//
```

Figure 4-1 Sample DEFINE AIX

A VISION:Builder file definition of the alternate path must be created with a K specified in the record format entry. The alternate key must be in the root segment of the file and specified as the record key field. JCL must be provided that points to the alternate index path. [Figure 4-2](#) shows the root segment of a sample VISION:Builder file definition of an alternate path. The key field of the primary index definition is EMP-NO. The key field is DEPT in the alternate path definition.

DETAILED GLOSSARY BY NAME FOR FILE DEFINITION - ALTPATH												
FILE IDENTIFICATION =						RECORD FORMAT = KEY SEQUENCED VSAM						
NUMBER OF SEGMENTS IN FILE = 1						RECORD SIZE = 4080						
NUMBER OF FIELDS IN FILE = 8						BUFFER SIZE = 4080						

* SEGMENT 1, LEVEL 1 *												
* SEGMENT NAME = EMPLOYEE *												

SEGMENT OCCURS N TIMES = 1						KEY FIELD 1 = DEPT			TYPE = C LENGTH = 3			
SEGMENT SIZE = 41												
NUMBER OF FIELDS IN SEGMENT = 8												
SEGMENT KEY CHARACTERISTICS = U												

FIELD NAME	FIELD TYPE	FIELD LOCATION	FIELD LENGTH	FIELD ENDING	DEC PLACES	CNT FOR	FIELD FOR	EDIT CODES () () ()	EDIT LENGTH	OUTPUT WIDTH	LINE NO	DATA BASE NAME/ COLUMN HEADING
DEPT	C	25	3						3	3		
EMP-NO	C	1	4						4	4		
FIRST	C	5	8						8	8	1	X
JOBCODE	C	28	1						1	1		
LAST	C	13	12						12	12		
NAME	C	5	20						20	20		
SALARY	P	29	4		2			(*) () ()	10	10		
SSN	C	33	9						9	9		

Figure 4-2 VISION:Builder File Definition of an Alternate Index Path

Alternate Index as a User File

The index of an alternate index path is automatically maintained by VSAM when the base cluster data file has been updated. If you want to access the alternate index and maintain or retrieve the index data yourself, you can do so using VISION:Builder.

If you want to access the alternate index, create a VISION:Builder file definition that maps the IBM configuration for an alternate index record. Also, VSAM ALT-INDEX must be specified in the record format entry of the File Definition window and the VSAM control interval size must be used as the buffer size.

Provide JCL that points to the alternate index. [Figure 4-3](#) illustrates a VISION:Builder file definition for the IBM alternate index record for a KSDS base with a key length of 3 bytes.

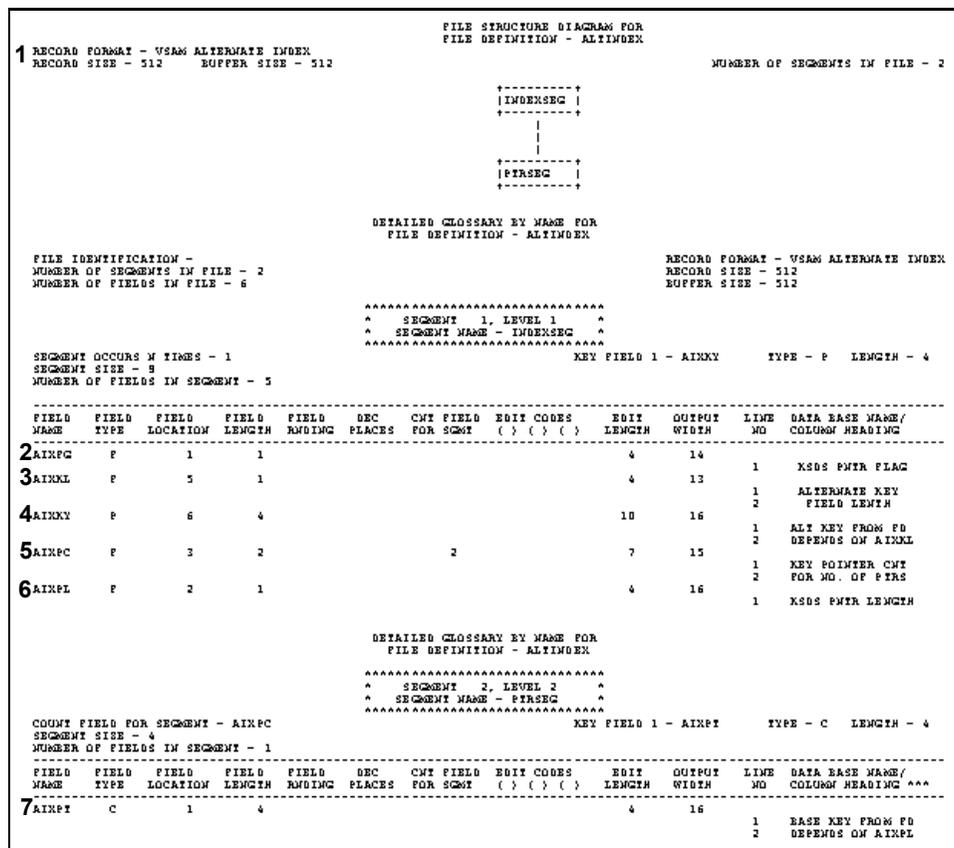


Figure 4-3 VISION:Builder File Definition of an Alternate Index for a KSDS Base

The following explanations refer to the numbered statements in [Figure 4-3](#).

Notes	Explanation
1	Record format is specified as VSAM ALT INDEX and buffer size is the CISIZE, which in this example is 512K.
2	Pointer flag: if 0, pointers are RBAs (ESDS base); if 1, pointers are keys (KSDS base).
3	Key field length: the alternate key length.
4	Key field: this is the alternate key. The length of this field depends on AIXKL.
5	Key pointer count: the number of pointers in this record.
6	Pointer length: if RBA pointer, the value of the length of this field is 4 bytes; if key pointer, the value of the length of this field is the key length of the KSDS base cluster.
7	Key field pointer to the base cluster data record. The length of this field depends on AIXPL.

Generic and Duplicate Keys

A generic key is the high order part of a record key whose length is less than the full key. Generic keys can be used for all key sequenced VSAM files (KSDS base, KSDS path, and ESDS path). They are defined in your VISION:Builder file definition as an overdefinition of the primary key.

Access to a VSAM file using a generic key is accomplished by using an additional file (RF) statement, with the generic key specified as the key field on the RF statement. Read operators (described in the [VISION:Builder Reference Guide](#)) are used in your VISION:Builder program to retrieve records based on the generic key.

The same read operators used for generic key search can also be used for retrieval of all occurrences of duplicate keys in an alternate index path for KSDS or ESDS. See the [VISION:Builder Reference Guide](#) for further information on the processing of generic and duplicate keys with VISION:Builder.

The term *own-code* refers to user supplied programming that can be integrated with VISION:Builder. Own-code might be required to provide functions that are not sufficiently general to warrant their inclusion in VISION:Builder. Own-code is an addition to VISION:Builder; it is your responsibility to guarantee the integrity of the system when it is used.

Own-code facilities provide you with two methods of specialized control in a VISION:Builder program: at specific points in VISION:Builder processing or for handling I/O functions for any specific file or files in the run.

The specific points in the VISION:Builder processing routines that allow access to user written modules are called own-code hooks and are identified by two-character identifiers. Own-code hooks are invoked by using the own-code (OC) statement. When an own-code hook is invoked, VISION:Builder transfers control to a user written routine. In any one run, many own-code hooks can be invoked, but VISION:Builder always transfers control to the same user routine known as your own-code control module (M4OWN).

M4OWN is a user written interface between VISION:Builder and the various own-code routines required for your processing. VISION:Builder provides the identity of the hook and a parameter list to M4OWN. It is the responsibility of M4OWN to locate the appropriate user routine and ensure that it is in main storage. The M4OWN routine is resident in storage while VISION:Builder is executing.

User handling of I/O functions is provided by the user I/O feature which allows you to control all I/O for a specific file without requiring user control of all files in the run. User I/O is available only when own-code is included in the VISION:Builder system.

You establish control of I/O functions for a file in an additional file (RF) statement. A user I/O module name is entered to identify the routine you have coded to control the I/O for the file. Whenever VISION:Builder attempts to perform an I/O operation for that file, control is transferred to the user written module along with parameter lists that contain all information pertinent to the file and its processing.

Integrating M4OWN

M4OWN can be integrated in one of two ways: static integration, where M4OWN is link edited with VISION:Builder, and dynamic integration, where M4OWN is loaded in main storage using the LOAD macro. M4OWN is resident in storage through the entire VISION:Builder run.

Static Integration M4OWN must be a load or object module in a partitioned data set. Both the CSECT name and the name of the member must be M4OWN. To use Static Integration, you must perform the Static Own Code Integration Relink step as described in the Installation instructions.

In the case of static integration, own-code receives control before VISION:Builder does any processing. This is accomplished through own-code hook 10 for the purpose of initializing various parameters.

Own-code hook 10 can also be used to activate hooks that can be activated only by this method and not using the OC statement. Refer to [Own-Code Hook 10 on page 5-9](#) for a complete description of these hooks.

Dynamic Integration Dynamic integration is performed whenever the first OC statement is encountered. VISION:Builder uses the LOAD macro to obtain the module whose name is provided on the OC statement. In this case, the module does not have to be called M4OWN or have a CSECT of this name. For simplicity, this description continues to use M4OWN as the name of the control module.

The module that is loaded must reside as a load module on a partitioned data set. The data set can be either SYS1.LINKLIB, the current JOBLIB, or another data set referenced by an M4OWN DD statement.

If an M4OWN module has been included with VISION:Builder during the link edit (static integration), dynamic integration of an own-code routine can be performed by setting an appropriate parameter at the hook 10 exit. This procedure inhibits static integration for the duration of the run.

Obtaining Space for Own-Code Routines

M4OWN is an interface routine between VISION:Builder and own-code. As such, it may require main storage to use as work space or to load other routines. This main storage can be obtained using GETMAIN at own-code hook 10 or 11.

Main storage can be obtained other than at own-code hook 10 or 11 using one of the following methods:

- | | |
|----------------|--|
| Permanent | M4OWN can be of such a size that it contains buffers or work areas large enough to accommodate any storage requirement imposed by the run. In this case, the space is permanently sacrificed for all VISION:Builder runs using own-code. |
| Semi-Permanent | <p>The semi-permanent method allows M4OWN to allocate storage for routines dynamically. This method relies on passing control to own-code at certain points solely for the purpose of allocating and freeing storage. These points are:</p> <ul style="list-style-type: none"> ■ Initially (own-code hook 51). ■ Just after decoding requests and before entering processing and passing the master file (own-code hook 61). |

When you request storage for own-code from VISION:Builder, the storage is awarded from the lower end of the range (that is, starting at the storage address that is passed to M4OWN when the hook is executed).

- | | |
|-----------|--|
| Temporary | The temporary method is an extension of the semi-permanent method. In this method, M4OWN can obtain storage dynamically for the M4OWN processing routine. The storage is obtained from storage allocated to VISION:Builder. In this situation, the storage belongs to own-code only while own-code actually has control. |
|-----------|--|

When M4OWN returns control to VISION:Builder, the storage is used by VISION:Builder. In this situation, the absolute origin of such storage can vary from call to call.

The origin and amount of available storage are specified in the primary calling list (see [Figure 5-2 on page 5-5](#)). This method is advantageous in that it does not rob VISION:Builder of excessive amounts of storage.

Since it is immaterial to VISION:Builder which methods are employed, VISION:Builder provides parameters so that any combination of the methods can be used.

Interrupts and Linkage Considerations

In transferring to own-code, VISION:Builder enables all programmable interrupts. Your own-code module must field any interrupts. If the own-code routine does not handle interrupts and an interrupt occurs, VISION:Builder terminates with an appropriate message. A user 60 ABEND is also issued.

Own-code routines can process their own interrupts by use of the SPIE macro; however, it is imperative that own-code restore the VISION:Builder PICA before returning to VISION:Builder. [Figure 5-1](#) shows an example of restoring the PICA.

Note: If you use 31-bit addressing, you must use the ESPIE macro in place of the SPIE macro.

<pre> . . . ESPIE SET, FIXUP, (8), PARAM=LIST1 ST 1, HOLD . . . L 5, HOLD ESPIE RESET, (5) . . . HOLD DC F'0' LIST1 DC A </pre>	<p>Provide exit routine for fixed-point overflow Save address returned in register 1</p> <p>Reload returned address Use execute form and old PICA address</p> <p>Parameter List</p>
--	--

Figure 5-1 Restoring the VISION:Builder PICA

VISION:Builder saves and restores all general registers used before and after transfer to M4OWN. If you use SAVE and RETURN macros, all registers are stored twice. VISION:Builder does NOT save floating point registers. These are the responsibility of M4OWN. The following register conventions are used:

- Register 1 This register is used to pass the address of a parameter list to M4OWN.
- Register 13 This register contains the address of an area set aside by VISION:Builder where M4OWN can save registers.
- Register 14 This register contains the return point to VISION:Builder. You should return with BR 14.
- Register 15 This register contains the entry point of M4OWN. It can be used by M4OWN as a base register. This register can also be used by M4OWN to pass a return code back to VISION:Builder. The meaning of the return code varies according to the hook.

The parameters passed to own-code are contained in a non-contiguous list. The two parts of this list are known as the primary calling list and the secondary calling list. Register 1 contains the address of the primary calling list. This list always exists, is fixed, five words long, and starts on a fullword boundary. The primary calling list is shown in [Figure 5-2](#). The secondary calling list exists only if word 5 of the primary list is non-zero. The secondary calling list is variable and dependent upon the hook invoked. Its format, where it exists, is documented with the description of the specific hook.

		← 2 Bytes →	← 2 Bytes →
Word 1	0	Own-Code Hook Identifier	Reserved
Word 2	4	Base of Available Storage for User Routines	
Word 3	8	Amount of Available Storage	
Word 4	12	Pointer to OWN Flag	
Word 5	16	Pointer to Secondary Calling List	

Figure 5-2 Primary Calling List

Own-code Hook Identifier	The two EBCDIC characters that identify the own code hook.
Base of Available Storage	The location of the first byte of available storage.
Amount of Available Storage	A binary fullword that contains the amount of storage that you can use.
Pointer to OWN Flag	The OWN flag is a 16-byte character string. This flag can be referenced in VISION:Builder requests by using the qualifier F and the name OWN. The flag is initially blank and is not subsequently used by VISION:Builder. It is provided for communication between M4OWN and the requests.
Pointer to Secondary Calling List	A pointer to the remainder of the list. The remaining portions of the list are dependent upon the particular hook. If there are no further parameters, this word is zero.

Own-Code Hook Naming Conventions

Type	Comments
1x	Mandatory for all own-code users. When own-code is used, these hooks are always activated and cannot be inhibited. They cannot be invoked from an OC statement.
2x	Can only be invoked from own-code hook 10.
3x	Hooks relevant during the entire VISION:Builder run.
5x	Hooks relevant for request decoding.

Type	Comments
6x	Hooks relevant for transaction processing (if VISION:Two, prior to request processing).
7x	Hooks relevant for request processing.
9x	Hooks relevant to I/O situations.

These classifications are of a general nature since it is not always possible to define the exact classification for certain hooks. See [Own-Code Hook Descriptions on page 5-9](#) for details.

Own-code hooks are documented as illustrated in [Figure 5-3](#).

Hook ID =

Time When Control is Passed to M4OWN
(In external terms.)

Secondary Calling List
(Parameters peculiar to the hooks that are provided by VISION:Builder to the own-code routine.)

Return Parameters
(Parameters returned to VISION:Builder by the own-code routine.)

Comments

Figure 5-3 Own-Code Hook Description Format

VISION:Builder Own-Code Hook Flow

Figure 5-4 depicts the points at which own-code hooks can be activated during either a VISION:Builder report step or processing step. Hook 30, when invoked, is active just before printing any VISION:Builder message.

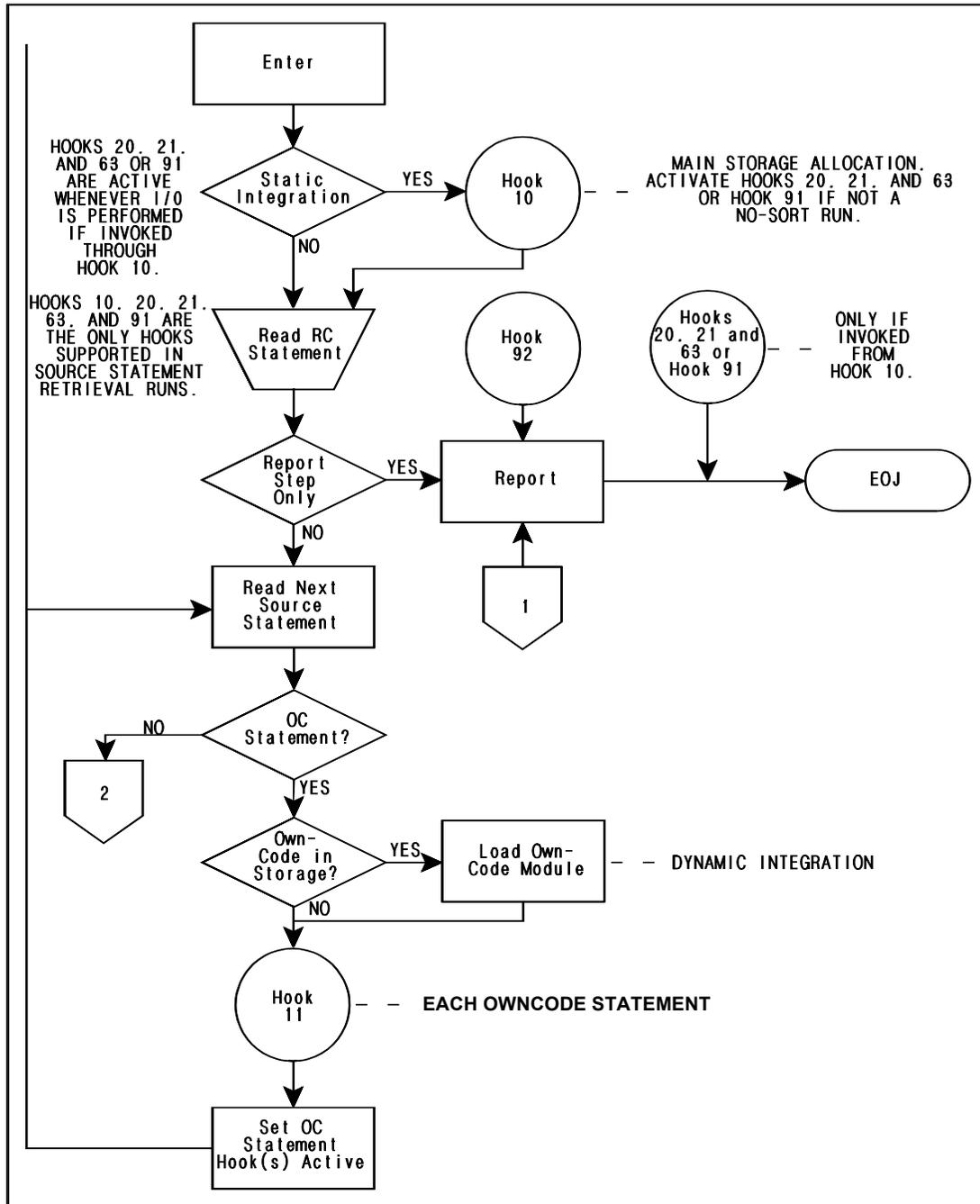


Figure 5-4 VISION:Builder Own-Code Flow (Page 1 of 2)

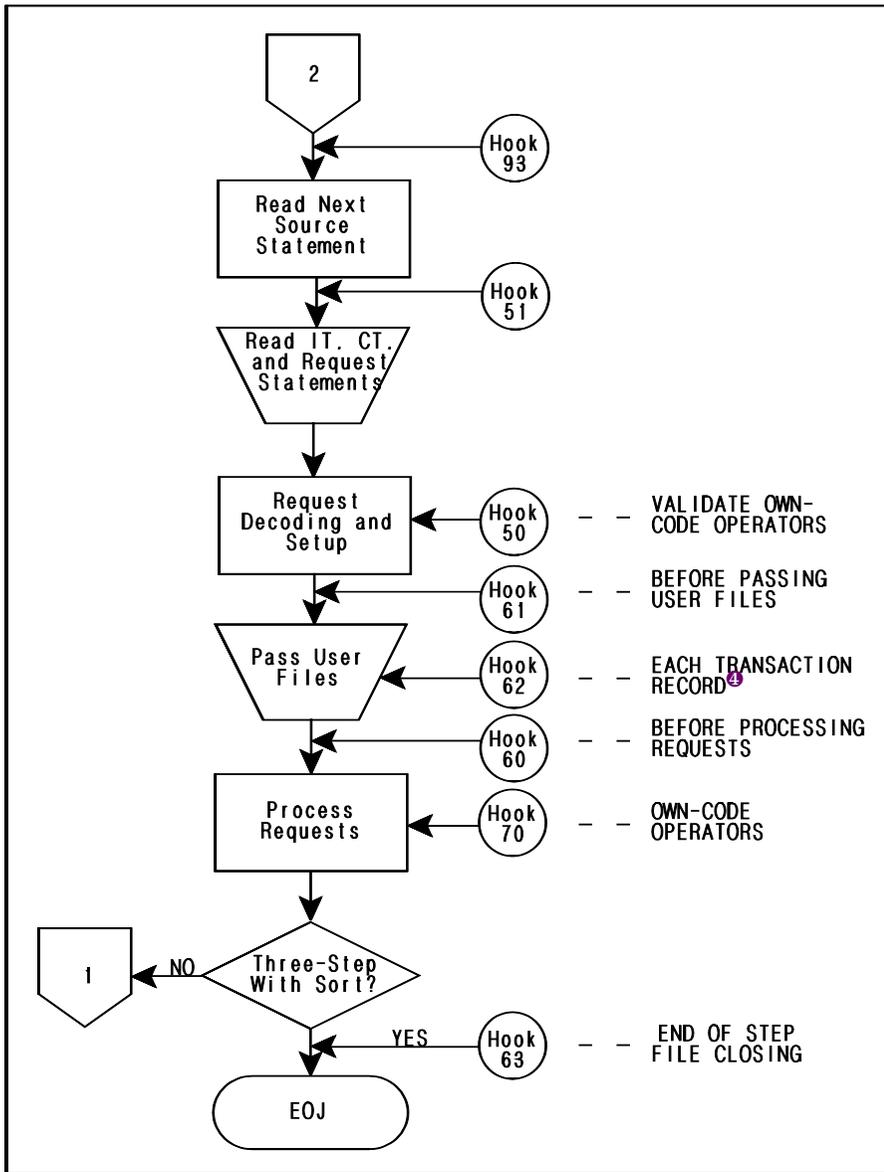


Figure 5-4 VISION:Builder Own-Code Flow (Page 2 of 2)

Own-Code Hook Descriptions

A detailed description of each hook follows.

Own-Code Hook 10

Hook ID = 10

Time When Control is Passed to M4OWN: Before any VISION:Builder processing occurs.

This hook is always active when static integration is used. Static integration can be inhibited at Hook 10 in order that dynamic integration can be used.

Secondary Calling List

	← 1 Byte →	← 3 Bytes →
0	FLAGS	Pointer to PARM Information
4	X'00'	Address of M4OWN

Note: The Pointer to PARM Information¹ contains the contents of Register 1 received by VISION:Builder when initially invoked. For example, if DL/I is the actual program named on the EXEC statement, this is the address of the parameter list passed from DL/I to VISION:Builder.

Return Parameters

FLAGS can be set as follows:

- X'80' User I/O. This flag indicates that the user will perform all I/O operations: OPEN, CLOSE, GET, and PUT. It automatically activates hooks 20, 21, and 63. OPENs must be performed at the first GET or PUT exit (hooks 20, 21) and CLOSEs must be performed at the EOJ exit (hook 63).
- X'40' User I/O error control. You receive control of all I/O errors that occur during sequential file processing. It automatically activates hook 91. In this case, VISION:Builder is performing all I/O and you process all I/O errors.

Note: These flags are mutually exclusive and must not be set at the same time.

The address of M4OWN in the secondary calling list can be modified at hook 10 to change the entry point of M4OWN for all other hooks or to inhibit static integration. Static integration is inhibited by storing hexadecimal zero in the second 4 bytes of the secondary calling list (address of M4OWN).

Comments

- Use this hook to do any initial GETMAINs required by own-code.
- Use this hook to obtain control of all I/O (for example, to process user labels).
- Use this hook to pass PARM information. You can use any PARM information you want. However, if the first two characters are M4, VISION:Builder assumes that the first eight characters specify a VISION:Builder parameter. Therefore, you should not use M4 as the first two characters of this PARM information.
- If user I/O is requested, the following is not allowed:
 - User I/O error handling (own-code hook 91).
 - No Sort runs (A Y for sort control on the RC statement).
- Use this hook to inhibit static integration. When static integration is inhibited, dynamic integration can be employed.
- Use this hook to change the M4OWN entry point.

Own-Code Hook 11

Hook ID = 11

Time When Control is Passed to M4OWN: After reading each OC statement, but before scanning the statement.

This hook is automatically activated when an OC statement is encountered. This hook cannot be inhibited.

Secondary Calling List

Own-code statement (the actual 80-byte image of the OC statement).

Return Parameters

None.

Comments

- Use this hook to manipulate the own-code statement as desired.
- You can perform GETMAINs at this hook.

Own-Code Hook 20

Hook ID = 20

Time When Control is Passed to M4OWN: Each GET.

Secondary Calling List

	← 1 Byte →	← 3 Bytes →
0	File Code	DCB Address
4	Not used – Reserved for VISION:Builder	
8	Input Record Length	

The File Code is as follows:

Code	File	Code	File
X'01'	Source Input (M4INPUT)	X'04'	Old Master File
X'06'	Transaction File ^④	X'0A'	Report File In
X'0E'	Coordinated File 1	X'0F'	Coordinated File 2
X'10'	Coordinated File 3	X'1E'	Coordinated File 4
X'1F'	Coordinated File 5	X'20'	Coordinated File 6
X'21'	Coordinated File 7	X'22'	Coordinated File 8
X'23'	Coordinated File 9		

Return Parameters

The address of the input record, as returned by IOCS, must be returned in Register 15. When end of file is reached, this register must contain 0. When you process undefined format records, the length of the input record must be returned in the appropriate field in the secondary calling list.

Comments

- You are responsible for all GET I/O except for the common library. At the first hook 20 exit for each file, you must do the following:
 1. Insert your own end of file exit address in the DCB. If you want I/O error control, you must alter the DCB accordingly.
 2. OPEN the file for input. User or non-standard label processing can be performed at this time.
 3. VISION:Builder builds a DCB for the file. I/O areas are allocated. You can use this DCB or use your own. The same is true of the I/O areas that can be obtained from the DCB. Own-code can perform any operation; however, the return to VISION:Builder must effectively simulate the return from the operating system IOCS.
 4. DD overrides are not permitted if the VISION:Builder DCB is used.
 5. You should save the DCB address so that, if necessary, you can CLOSE it at hook 63.

- Hook 20 is not invoked for VSAM files or database files.

Own-Code Hook 21

Hook ID = 21

Time When Control is Passed to M4OWN: Each PUT.

Secondary Calling List

	← 1 Byte →	← 3 Bytes →
0	File Code	DCB Address
4	Work Area Address	
8	Output Record Length	

Code	File	Code	File
X'02'	Source Listing (M4LIST)	X'03'	Sort Statements
X'05'	New Master File ^④	X'07'	Audit File ^④
X'08'	Rejected Transactions File ^④	X'09'	Report File Out
X'0B'	Report File Out 2	X'0C'	Report File Out 3

Code	File	Code	File
X'0D'	Report File Out 4	X'24'	Report File Out 5
X'25'	Report File Out 6	X'26'	Report File Out 7
X'27'	Report File Out 8	X'28'	Report File Out 9
X'13'	Source Statement Output File	X'14'	Subfile 1
X'15'	Subfile 2	X'16'	Subfile 3
X'17'	Subfile 4	X'18'	Subfile 5
X'19'	Subfile 6	X'1A'	Subfile 7
X'1B'	Subfile 8	X'1C'	Subfile 9
X'1D'	Subfile 0	X'29'	Alternate Lists (M4LIST1)
X'2D'	Program Analyzer (PAL)	X'30'	Report Summary File

Return Parameters

None.

Comments

- You are responsible for all PUT I/O except for the common library. You must OPEN the file for output at the first hook 21 exit.
- If you want I/O error control, you must alter the DCB accordingly.
- OPEN the file for output. User or non-standard label processing can be performed at this time.
- VISION:Builder builds a DCB for the file. I/O areas are allocated. You can use this DCB or use your own. The same is true of the I/O areas that can be obtained from the DCBs. Own-code can perform any operation; however, the return to VISION:Builder must effectively simulate the return from the operating system IOCS.
- DD overrides are not permitted if the VISION:Builder DCB are used.
- You must save the DCB address so that, if necessary, you can CLOSE the file at hook 63.
- The length of the record to be PUT is available in the secondary calling list.
- Hook 21 is not invoked for VSAM files or database files.

Own-Code Hook 30

Hook ID = 30

Time When Control is Passed to M4OWN: Just before printing a message.

Secondary Calling List

	← 1 Byte →	← 3 Bytes →
0	FLAGS	Message Address
4	Code Letters	Message Number

Return Parameters

If Register 15 = 0, the message is written.

If Register 15 = 4, the message is suppressed.

Comments

- Flag settings are as follows:
 - X'01' — Terminal message.
 - X'02' — Continuation of a previous message.
 - X'04' — Message is to be printed on console.
 - X'08' — Message is to be printed on printer.
- Code letters and message number are in EBCDIC format.
- You can alter VISION:Builder messages or supply your own.
- VISION:Builder message format is as follows:

Byte	Description	Byte	Description
1	ASA control character	2-3	Asterisks
4	Blank	5-7	Component code letters (CLS=library, MK4=VISION:Builder)
8-11	Message code letters	12-13	Blanks
14-17	The word TYPE	18	Blank
19	Type code	20-21	Blanks
22-121	Message text		

Own-Code Hook 50

Hook ID = 50

Time When Control is Passed to M4OWN When an own-code operator is to be validated during the decoding of a request.

Secondary Calling List

The PR statement (the actual 80-byte image of the PR statement).

Note: Positions 1-8 of this statement are used for VISION:Builder internal parameters and must not be altered by own-code.

Return Parameters

If Register 15 = 0, the operator has been verified as valid.

If Register 15 = 4, the operator is not valid.

Comments

- Own-code hook 50 must be used to validate own-code operators (PR statement “operation” field). If it is not used, the own-code operators are assumed to be illegal.
- Own-code hook 50 must be used in conjunction with own-code hook 70.
- A result field (Operand C) must always be provided for own-code operators.
- Own-code operators may be any single character (right-adjusted or left-adjusted in the PR statement operation field) with the exception of the following:
 - + (plus)
 - (minus)
 - / (virgule)
 - _ (underline)
 - | (bar)
 - * (asterisk)
 - I, J, K, R
 - 0 to 9

Any EBCDIC combination that produces internal representations of X'FA' to X'FF' or X'00' to X'09'.

Own-Code Hook 51

Hook ID = 51

Time When Control is Passed to M4OWN: After the run control group and CT statements have been read, but before decoding requests.

Secondary Calling List

None.

Return Parameters

Register 15 contains the amount of storage obtained from VISION:Builder.

Comments

To obtain virtual storage from VISION:Builder (semi-permanent).

Own-Code Hook 60

Hook ID = 60

Time When Control is Passed to M4OWN: After all transactions (if VISION:Two, prior to request processing) have been processed for a record, but before processing normal (type N) requests.

Secondary Calling List

	← 4 Bytes →
0	Address of Old Master File Record
4	Address of New Master File Record

Return Parameters

If Register 15 = 0, request processing continues normally.

If Register 15 = 4, all request processing of normal (type N) requests for this record are bypassed.

Comments

None.

Own-Code Hook 61

Hook ID = 61

Time When Control is Passed to M4OWN: After request decoding, but before passing any files.

Secondary Calling List

None.

Return Parameters

Register 15 contains the amount of storage obtained from VISION:Builder.

Comments

- Use this hook to obtain storage from VISION:Builder (semi-permanent).
- Storage cannot be returned to VISION:Builder.

Own-Code Hook 62

Hook ID = 62 ⁴

Time When Control is Passed to M4OWN: After reading each transaction record.

Secondary Calling List

The transaction record.

Return Parameters

If Register 15 = 0, transaction processing continues normally.

If Register 15 = 4, the transaction is rejected.

Comments

A transaction rejected in this manner is output to the rejected transaction file (M4REJECT) but is not reflected in the setting of the TRAN flag.

Own-Code Hook 63

Hook ID = 63

Time When Control is Passed to M4OWN: End of job, before returning to the system.

Secondary Calling List

Halfword condition code with which VISION:Builder returns. This can be modified in order to change the condition code returned by VISION:Builder. The value in the CONDCODE flag has already been added to the condition code when hook 63 receives control.

Return Parameters

The condition code in the secondary calling list can be modified by M4OWN. This code is returned to the calling program when VISION:Builder exits. If the operating system called VISION:Builder, this condition code can be tested by the JCL in subsequent steps.

Comments

Use this hook to pass along instructions for the operator or timing.

- If hooks 20 or 21 have been invoked using hook 10, this hook is automatically activated to allow you to CLOSE all files. You must have saved the DCB address of each file at hook 20 or 21 or have some other way of determining the DCB addresses in order to properly close each file.

Own-Code Hook 70

Hook ID = 70

Time When Control is Passed to M4OWN: When an own-code operator is to be executed during the processing of a request.

Secondary Calling List

0	Type of Field 1	Address of Field 1		
4	Type of Field 2	Address of Field 2		
8	Type of Field 3	Address of Field 3		
12	Length of Field 1	Scale of Field 1	Length of Field 2	Scale of Field 2
16	Length of Field 3	Scale of Field 3	Operation Code	Rounding Indicator
20	Number of Fields	VISION:Builder Internal Use		
	← 1 Byte →	← 1 Byte →	← 1 Byte →	← 1 Byte →

Field Type	Code	Field Type	Code
Character	X'00'	Zoned	X'04'
Packed	X'08'	Fixed point	X'0C'
Floating point	X'10'		

The lengths of fields are supplied as Length -1. The scale factor is the number of implied decimal places and is required for decimal point adjustment. These parameters are fixed point binary.

Operation Code:

Any single EBCDIC character other than the following:

- + (plus)
- - (minus)
- / (virgule)
- _ (underline)
- | (bar)
- * (asterisk)
- I, J, K, R
- 0 to 9
- Internal codes X'FA' to X'FF' or X'00' to X'09'

Fields stored in the secondary calling list have the following relationships to the input statement (result is always field 3):

- One field is field 3 (result of the input statement).
- Two fields are fields 1 and 3 (operand A or B and result of the input statement).
- Three fields are fields 1, 2, and 3 (operand A, operand B, and result).

Rounding Indicator

If the code is X'00', the result field does not have the rounding attribute.

If the code is X'04', the result field has the rounding attribute.

Return Parameters

If Register 15 = 0, the data has been stored and is valid.

If Register 15 = 4, the result field must be treated as invalid.

Comments

- If a field other than the result field is invalid or missing, control is not passed, and the result field is flagged as invalid.
- Own-code hook 70 must be used in conjunction with own-code hook 50.
- A result field must always be provided for own-code operators.
- If the result field is missing, control is not passed.

Own-Code Hook 91

Hook ID = 91

Time When Control is Passed to M4OWN: Whenever an I/O error occurs.

Secondary Calling List

0	CCW Displacement	Address of Status Indicator	
4	Status Information	DCB Address	
8	0	File Code	Reserved
	← 1 Byte →	← 1 Byte →	← 2 Bytes →

Bytes 0-7 are the contents of registers 0 and 1 respectively, upon entry to the SYNAD routine. The file code is defined as follows:

Code	File	Code	File
X'01'	Source Input (M4INPUT)	X'02'	Source Listing (M4LIST)
X'03'	Sort Statements	X'04'	Old Master File
X'05'	New Master File ⁴	X'06'	Transaction File ⁴
X'07'	Audit File ⁴	X'08'	Rejected Transactions File ⁴
X'09'	Report File Out	X'0A'	Report File In
X'0B'	Report File Out 2	X'0C'	Report File Out 3
X'0D'	Report File Out 4	X'24'	Report File Out 5
X'25'	Report File Out 6	X'26'	Report File Out 7
X'27'	Report File Out 8	X'28'	Report File Out 9
X'0E'	Coordinated File 1	X'0F'	Coordinated File 2

Code	File	Code	File
X'10'	Coordinated File 3	X'1E'	Coordinated File 4
X'1F'	Coordinated File 5	X'20'	Coordinated File 6
X'21'	Coordinated File 7	X'22'	Coordinated File 8
X'23'	Coordinated File 9	X'13'	Source Statement Output File
X'14'	Subfile 1	X'15'	Subfile 2
X'16'	Subfile 3	X'17'	Subfile 4
X'18'	Subfile 5	X'19'	Subfile 6
X'1A'	Subfile 7	X'1B'	Subfile 8
X'1C'	Subfile 9	X'1D'	Subfile 0
X'29'	Alternate Lists (M4LIST1)	X'2D'	Program Analyzer (PAL)
X'30'	Report Summary File		

Return Parameters

None.

Comments

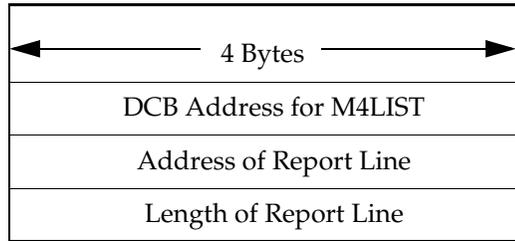
- You are responsible for run termination. Processing continues if control is returned to VISION:Builder. The block is skipped.
- If this hook is invoked using the OC statement, it does not apply to I/O errors from source input, source listing, or the VISION:Builder report file. These files are included if the hook is invoked through hook 10.
- Note that I/O errors from the common library are not passed to you. These errors always terminate a run.
- Hook 91 is not invoked for VSAM files or database files.

Own-Code Hook 92

Hook ID = 92

Time When Control is Passed to M4OWN: During single-step runs only and prior to writing each formatted text line to the printer.

Secondary Calling List



Return Parameters

If Register 15 = 0, the report line is output.

If Register 15 = 4, the report line is not output.

Comments

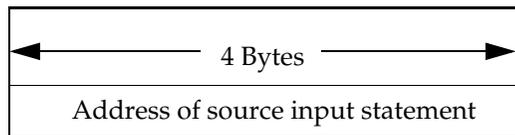
- This hook provides access to the formatted report line for modification.
- Optionally, this hook allows the user to issue all PUTs, suppressing those issued by VISION:Builder.
- Blank lines may not be passed to the user routing individually since VISION:Builder optimizes them using carriage control.
- VISION:Builder keeps track of the number of system-generated lines and uses this number to determine when to issue page ejects. Therefore, care must be taken when the own-code routine inserts physical lines into the page which could cause incorrect page formatting.

Own-Code Hook 93

Hook ID = 93

Time When Control is Passed to M4OWN: After reading each source input statement, but before processing.

Secondary Calling List



Return Parameters

None.

Comments

- Provides access to the 80-column VISION:Builder source input statement.
- Allows the user to perform editing on the source statements (for example, column headings, titles, and PR statements).

Variable Length Fields with Own-Code

The secondary calling list does not contain the address of a variable length (type V) field used as an operand of an own-code operator. For non-temporary fields, the address passed is the first variable length field in the segment. For temporary fields, the address passed is a word that contains the address of the variable length field.

User I/O

The user I/O feature of own-code, available only when own-code is included in the VISION:Builder system, allows you to control all I/O for a specific file without requiring that all I/O for all files be controlled. User I/O can be specified for any or all of the files listed in the table on page [5-24](#), but user I/O cannot be specified when own-code I/O using hooks 20, 21, or 91 is specified.

Interrupt handling is the same with user I/O as with existing own-code modules. Whenever VISION:Builder attempts I/O for a file where user I/O is invoked, control is passed to a module written by you. VISION:Builder passes a primary parameter list and a series of secondary parameter lists to your module.

The primary parameter list and some of the fields in the secondary parameter lists are all that is necessary for a basic interface with most languages. The secondary parameter lists contain additional items and addresses of other lists that may be manipulated by assembler language if you want to code a more general interface.

You specify user controlled I/O for a file using an RF statement containing the following entries:

- | | | |
|------------|------------------|------------------------|
| ■ Run name | ■ Form code (RF) | ■ User file name |
| ■ Label | ■ DTF/DD name | ■ User I/O module name |

The table on page [5-24](#) indicates the files where user I/O can be specified plus the requirements for the user file name entry. The user I/O module name is the link edited module name of the routine that you have coded to control the I/O for the file. If the same module is specified for more than one file, a separate copy of the module is loaded for each file unless the module was link edited reentrant.

On entry to your I/O module, register 1 contains the address of the primary parameter list, register 14 contains the return address, and register 15 contains the address of your I/O module. Register 13 contains the address of an area of eighteen fullwords in which you can save registers. You do not need to save or restore the registers since VISION:Builder saves and restores all registers, using a different area, before and after transfer to your I/O module.

ddname or equivalent (if overridden in OS/390)	User File Name
M4OLD M4NEW ⁴ MVAUDIT ⁴	The entry must match the file name entered in the RC statement.
M4TRAN ⁴ M4REJCT ⁴	If RF statements are entered for both M4TRAN and M4REJCT, the entries in user file name must match.
M4CORDn	User file name required (n = 1-9).
M4SUBFn	User file name required (n = 1-9).

Primary Parameter List

The primary parameter list is a series of fullword addresses of tables and data areas; each table or data area begins on a doubleword boundary. The last of these addresses has a hexadecimal 80 in the high order byte. [Figure 5-5](#) illustrates this parameter list.

Byte	Fullword	
0	COMMUNICATION TABLE ADDRESS	
4	FILE TABLE ADDRESS	
8	INPUT RECORD ADDRESS	
12	OUTPUT RECORD ADDRESS	
16	KEY ADDRESS	
20	OWN-CODE FLAG ADDRESS	
24	X'80'	PARAMETER ADDRESS

Figure 5-5 Primary Parameter List

The following table contains an explanation of each of the addresses in the primary parameter list.

Address	Description
COMMUNICATIONTABLE ADDRESS	The address of a table containing seven fullwords that are used for communication to and from your module.
FILE TABLE ADDRESS	The address of a table containing characteristics of the file that you are controlling.
INPUT RECORD ADDRESS	The address of the record to be input to VISION:Builder. If the file has no input associated with it, this address is zero. The address points to the beginning of the data of the record unless there is a record descriptor word; in that case, the address points to the 4-byte record descriptor word. This address is set either by VISION:Builder on return from an OPEN call if requested by you or by you on each GET function.
OUTPUT RECORD ADDRESS	The address of the record to be output. If the file has no output associated with it, this address is 0. The address points to the beginning of the data of the record unless there is a record descriptor word. In that case, the address points to the 4-byte record descriptor word. This address is set by VISION:Builder on each add, replace, or delete function.
KEY ADDRESS	VISION:Builder uses the definition of the primary key for the file (if any) and places this key in a work area. The key address contains the address of this work area that contains the key of the current or requested record. In the case of sequential input, the key area contains the key of the last record processed.
OWN-CODE FLAG ADDRESS	The address of the 16-byte OWN flag provided for communication between M4OWN and requests.
PARAMETER ADDRESS	The address of OS PARM information as passed to VISION:Builder.

Communication Table

The communication table contains codes, an address, and values that VISION:Builder uses to communicate to and from your module. [Figure 5-6](#) is a representation of the communication table and its contents.

Byte	Fullword
0	FUNCTION CODE
4	STATUS CODE
8	TEMPORARY VIRTUAL STORAGE START
12	TEMPORARY VIRTUAL STORAGE LENGTH
16	VIRTUAL STORAGE TAKEN
20	INPUT RECORD DATA SIZE
24	OUTPUT RECORD DATA SIZE

Figure 5-6 Communication Table

The following table contains explanations of the entries in the communication table (Figure 5-6).

Entry	Description	
FUNCTION CODE	A hexadecimal, fullword, fixed point, encoded number indicating what function VISION:Builder wants your I/O module to perform. For function codes 00000014, 0000001C, and 00000020, for an update-in-place file when the file I/O is user controlled, VISION:Builder does not set X'FF' in the first byte of a record being deleted.	
Code	Meaning	Comments
00000004 ⁴	Open the file.	
00000008	Close the file.	
0000000C	Get a record sequentially.	
00000010 ⁴	Add a record sequentially	For records that are to be added to the file. This would be the function code for files that are only output.
00000014 ⁴	Delete record sequentially	For an update-in-place sequential access file.
00000018 ⁴	Replace record	For an update-in-place sequential access file, when a record has been read in and now is to be written out.
0000001C ⁴	Add a record sequentially	For outputting a newly created record to an update-in-place file.
00000020 ⁴	Delete a record by key	For an update-in-place random access file.
00000024 ⁴	Replace a record by key	For an update-in-place random access file, when a record has been read in and is now to be written out.
00000028 ⁴	Add record by key	For outputting a newly created record to a random access file being updated-in-place.
0000002C	Get a record by key	

Entry	Description
STATUS CODE	A hexadecimal, fullword, fixed point encoded number specifying your response to the VISION:Builder requested functions. Upon entry to your user I/O module, the status indicator is set to 0. Your I/O module must replace a number into the status code before it returns to VISION:Builder, depending on the results of the requested function. If an incompatible or unrecognized status code for the requested operation is returned in the status indicator, VISION:Builder issues a type 5 diagnostic message specifying the invalid code and terminates the run.
Code	Meaning
00000000	Requested function successfully completed. Since this code is the status indicator upon entry to the user I/O module, it is not necessary to set it to 0 if the function was successfully completed.
00000004	End of file has occurred on an X'0C' function.
00000008	Record not found during an X'2C' function.
0000000C	Request by you to VISION:Builder to allocate the record work area for the file and set the address of this area in the record address in the primary parameter list. VISION:Builder allocates an area equal to the record size (plus 4 if variable format) in the file table. This status code applies only on return from OPEN calls for input fields; output record areas are always allocated by VISION:Builder.
FFxxxxxx	Terminate the run. xxxxxx is a user-supplied hexadecimal value. VISION:Builder normally terminates the run, that is, it closes all of the fields in the run (user I/O modules are called with a close function) and issues a type 5 message specifying that you terminated the run.

Entry	Description
TEMPORARY VIRTUAL STORAGE START	<p>The address of available temporary work space. This area begins on a doubleword boundary. During all functions, except opening a file, this temporary storage may be used as a work space. The contents and the location of this area may be changed by VISION:Builder between calls to your I/O module. Under OS/390, this entry is unused during an open function and is 0. During this time, storage is available to you through the operating system using GETMAINs and LOADs.</p> <p>You can perform GETMAINs for any permanent work area or buffers that are required. Also, at this time, you can perform GETMAINs and FREEMAINs for any temporary work space.</p>
TEMPORARY VIRTUAL STORAGE LENGTH	<p>The amount of available temporary work space. Under OS/390, this entry is unused during an open function and is 0. This value should be used to determine if enough storage is available for your I/O module functions.</p>
VIRTUAL STORAGE TAKEN	<p>Not used under OS/390.</p>
INPUT RECORD DATA SIZE	<p>For input files, this is the size of the record just retrieved by your module. The value must be set by you at each GET function and does not include any record descriptor word (4 bytes), if one is present. For output only files, this word is unused.</p>
OUTPUT RECORD DATA SIZE	<p>For output files, this is the size of the record that VISION:Builder is requesting your module to output. This value is set by VISION:Builder at each add, replace, or delete function and does not include any record descriptor word (4 bytes) if one is present.</p>

File Table

The file table contains information about the file obtained from the file definition or from En/Rn statements. The format of the file table is shown in [Figure 5-7](#).

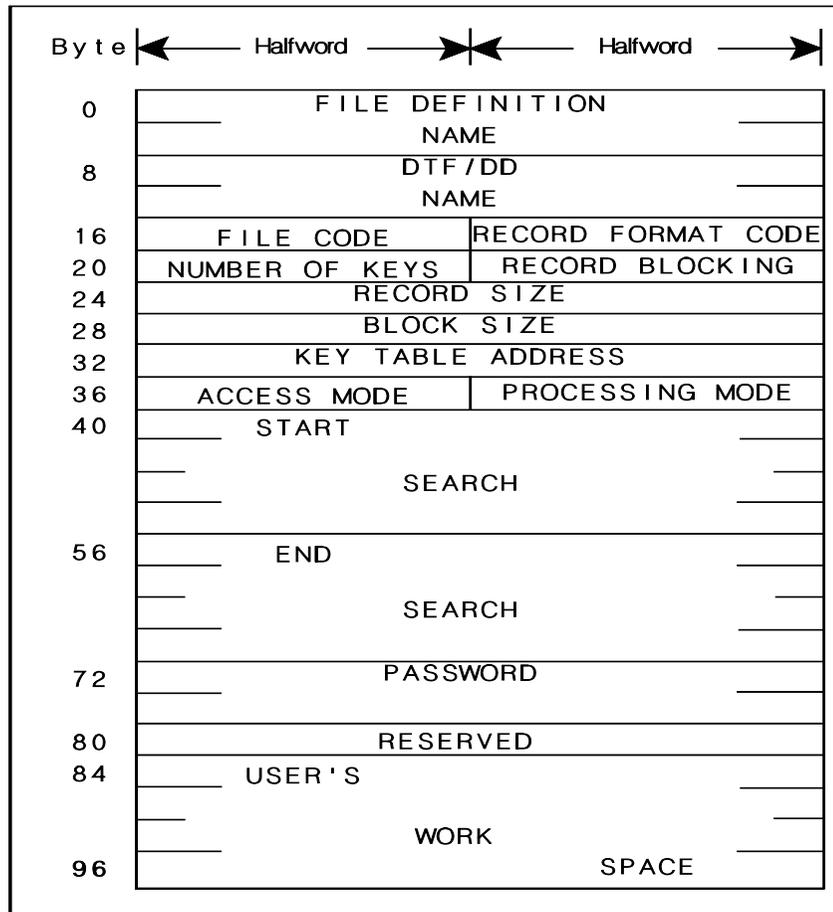


Figure 5-7 File Table

The following table contains explanations of the entries of the file table:

Entry	Description
FILE DEFINITION NAME	The name of a related file definition or the subfile name entered in the RF statement.
DTF/DD NAME	The 8-byte name entered in the RF statement.
FILE CODE	A 2-byte fixed point number that identifies the VISION:Builder file being controlled. This entry relates directly to the ddname as shown in the table on page 5-33 .

Entry	Description																				
RECORD FORMAT CODE	<p>A 2-byte, fixed point number that identifies the format of the file. The information is obtained from either an associated file definition or from En/Rn statements. The hexadecimal codes are:</p> <table border="1"> <thead> <tr> <th>Record Format Code</th> <th>Format</th> </tr> </thead> <tbody> <tr> <td>4</td> <td>F</td> </tr> <tr> <td>8</td> <td>V</td> </tr> <tr> <td>C</td> <td>U</td> </tr> <tr> <td>10</td> <td>I</td> </tr> <tr> <td>14</td> <td>J</td> </tr> <tr> <td>18</td> <td>P</td> </tr> <tr> <td>20</td> <td>A (VSAM Alternate Index)</td> </tr> <tr> <td>20</td> <td>K (Key-sequenced VSAM file)</td> </tr> <tr> <td>24</td> <td>E (Entry-sequenced VSAM file)</td> </tr> </tbody> </table> <p>The code 18 applies only to the extended file processing option, where you receive on output, and must supply on input, a record in the VISION:Builder packed format.</p>	Record Format Code	Format	4	F	8	V	C	U	10	I	14	J	18	P	20	A (VSAM Alternate Index)	20	K (Key-sequenced VSAM file)	24	E (Entry-sequenced VSAM file)
Record Format Code	Format																				
4	F																				
8	V																				
C	U																				
10	I																				
14	J																				
18	P																				
20	A (VSAM Alternate Index)																				
20	K (Key-sequenced VSAM file)																				
24	E (Entry-sequenced VSAM file)																				
NUMBER OF KEYS	A 2-byte, fixed point number. The information is obtained from either an associated file definition or from En/Rn statements.																				
RECORD BLOCKING	A 2-byte, hexadecimal code; 0004 indicates unblocked and 0008 indicates blocked records.																				
RECORD SIZE	A 4-byte, fixed point number. The information is obtained from either an associated file definition or from En/Rn statements. This is the size of the data part of the record; it does not include the record descriptor word (4 bytes), if one is present.																				
BLOCK SIZE	A 4-byte, fixed point number. The information is obtained from either an associated file definition or from En/Rn statements.																				

Entry	Description
KEY TABLE ADDRESS	<p>A pointer to a table containing information about the record keys (level one segment keys). If the table is not built, the pointer to the key table is binary zero. If the table is built, the information for M4OLD, M4NEW,⁴ M4CORDn, M4TRAN,⁴ M4AUDIT,⁴ and M4REJCT⁴ files is obtained from the related file definition; if there is no related file definition for M4TRAN⁴ or M4REJCT,⁴ no key table is built for these files. For an M4SUBFn sequential file, no key table is built. For an M4SUBFn ISAM file, the single key is obtained from the first EN/Rn set decoded that is to be output to the subfile. Within that Rn set it is the major sort key. If entire record selection is specified, it is the record key or if the above cannot be satisfied, it is the field name specified on the first Rn statement.</p>
ACCESS MODE	A 2-byte, hexadecimal code; 0004 indicates random and 0008 indicates sequential access.
PROCESSING MODE	A 2-byte, hexadecimal code; 0004 indicates output only, 0008 indicates input only, and 000C indicates update-in-place (input/output).
START SEARCH	The key value, left-aligned, exactly as specified on the RC statement for start search. This is the key of the record to begin processing. The initial value of the start search field is low value (X'00') for the entire 16 bytes of the field.

Entry	Description
END SEARCH	The key value, left-aligned, exactly as specified on the RC statement for end search. This is the key of the record after which processing should terminate. The initial value of the end search field is high value (X'FF') for the entire 16 bytes of the field.
PASSWORD	The password as supplied on the RF statement for VSAM files by you. It is left blank for all non-VSAM files.
USER'S WORK SPACE	Four fullwords for your own use, such as pointers to modules to handle special functions for a given file.

File Code (Decimal)	VISION:Builder ddname	File Code (Decimal)	VISION:Builder ddname
1	M4OLD	28	M4AUDIT ⁴
2	M4NEW ⁴	29	M4REJCT ⁴
3	M4TRAN ⁴	33	M4SUBF1
4	M4CORD1	34	M4SUBF2
5	M4CORD2	35	M4SUBF3
6	M4CORD3	36	M4SUBF4
7	M4CORD4	37	M4SUBF5
8	M4CORD5	38	M4SUBF6
9	M4CORD6	39	M4SUBF7
10	M4CORD7	40	M4SUBF8
11	M4CORD8	41	M4SUBF9
12	M4CORD9	42	M4SUBF0

Key Table

The key table contains information about the record keys (the keys of the level one segment). The table consists of a series of contiguous entries, each beginning on a fullword boundary, with one entry for each key. The entries are ordered from major key (key number 1) to minor key. The structure of the table is illustrated in [Figure 5-8 on page 5-35](#).

The following table contains explanations for the table entries:

Entry	Description												
TYPE CODE	A 2-byte, fixed point number encoding the field type of the key. The hexadecimal codes are: <table border="1" data-bbox="711 449 956 770"> <thead> <tr> <th>Code</th> <th>Type</th> </tr> </thead> <tbody> <tr> <td>4</td> <td>C</td> </tr> <tr> <td>8</td> <td>Z</td> </tr> <tr> <td>C</td> <td>P, L, or S</td> </tr> <tr> <td>10</td> <td>F</td> </tr> <tr> <td>14</td> <td>E</td> </tr> </tbody> </table>	Code	Type	4	C	8	Z	C	P, L, or S	10	F	14	E
Code	Type												
4	C												
8	Z												
C	P, L, or S												
10	F												
14	E												
SCALE	A 2-byte, fixed point number (0 to 9) describing the scaling factor according to the definition of the field.												
LENGTH	The defined length of the field, stored as a 2-byte, fixed point number.												
FLAGS	A 1-byte field containing eight 1-bit indicators as follows: <p>0 . . . Field does not have the rounding attribute.</p> <p>1 . . . Field has the rounding attribute.</p> <p>Only the first bit can be referenced; the other seven are reserved.</p>												
DISPLACEMENT INTO ROOT SEGMENT	A 4-byte, fixed point number that is the displacement into the record where the key field begins.												

Byte	← Halfword →		← Halfword →		Halfword
0	Key Field TYPE CODE	Key Field SCALE		Entry for First Key Field	
4	Key Field LENGTH	Flags	Reserved		
8	DISPLACEMENT INTO ROOT SEGMENT				
	Key Field TYPE CODE	Key Field SCALE		Entry for nth Key Field	
	Key Field LENGTH	Flags	Reserved		
	DISPLACEMENT INTO ROOT SEGMENT				

Figure 5-8 Key Table Structure

Relationships Among Tables

Figure 5-9 illustrates the various relationships among the tables which are passed to your I/O module.

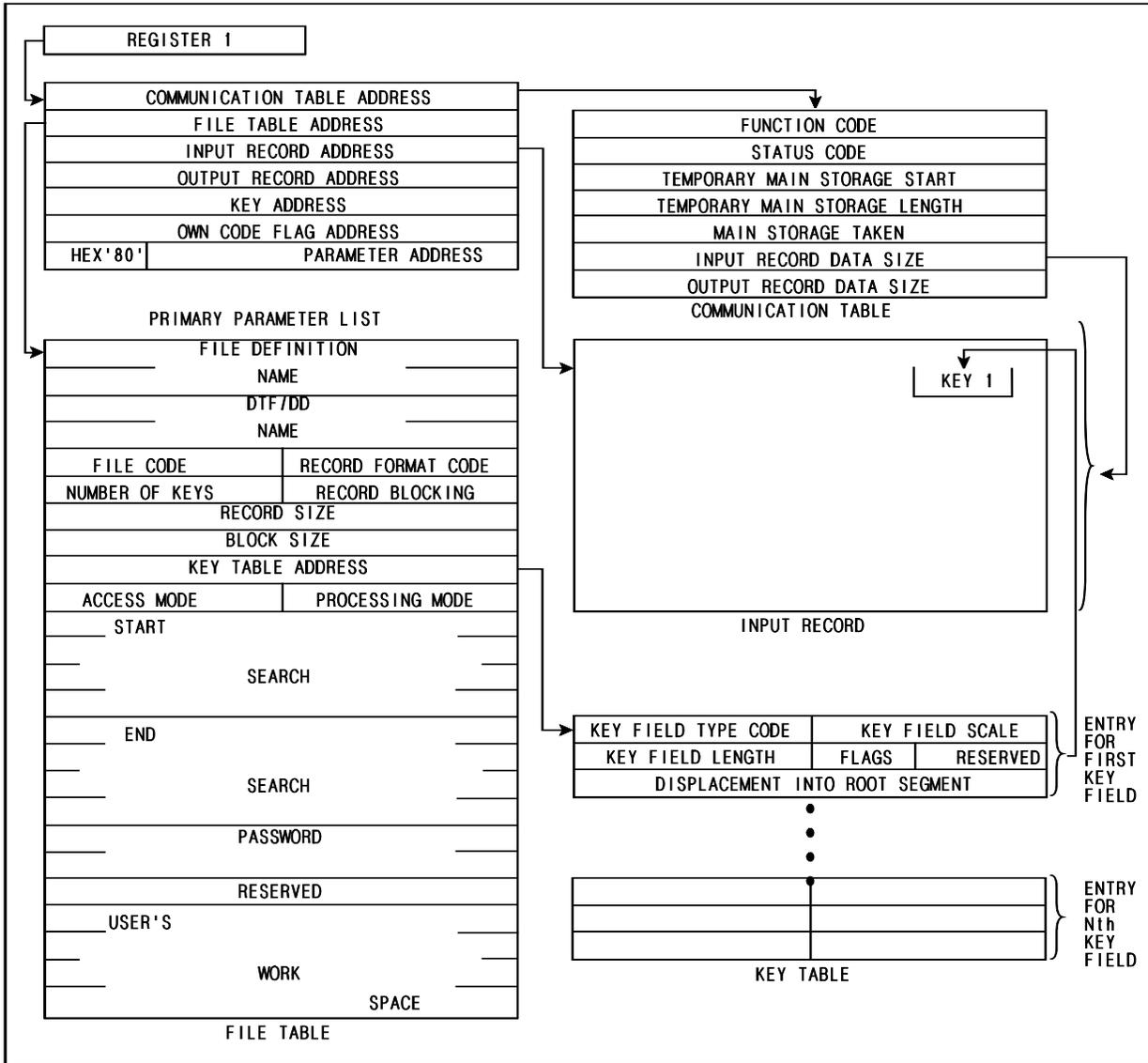


Figure 5-9 Table Relationships

Update-in-Place Example ⁴

The following is an example of the calls VISION:Builder would make to your I/O module if you are controlling the I/O for an update-in-place, randomly-accessed M4OLD file.

Master File Record Keys	Transaction File Record Keys	Attempted Action
A	G	Update
D	B	Update
G	A	Delete
J	C	Create

Function Code (Hexadecimal)	Key	Explanation	Returned Status Code
00000004	--	Open M4OLD	0
0000002C	G	Get record G	0
00000024	G	Replace record G	0
0000002C	B	Get record B. After this call, VISION:Builder outputs the transaction to M4REJCT	8 (not found)
0000002C	A	Get record A	0
00000020	A	Delete record A	0
0000002C	C	Get record C	8
00000028	C	Add record C	0
00000008	--	Close M4OLD	0

COBOL Example of User I/O

[Figure 5-10](#) illustrates a COBOL user I/O program that communicates to VISION:Builder through its linkage section. The COBOL example is retrieval only. The linkage section ends with the input record address.

```

ID DIVISION.
PROGRAM-ID.
AUTHOR.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MARKIV-FILE-IN ASSIGN TO UT-S-SYSIN.
DATA DIVISION.
FILE SECTION.
FD MARKIV-FILE-IN

```

Figure 5-10 COBOL User I/O (Page 1 of 3)

```

RECORDING MODE IS F
LABEL RECORDS ARE STANDARD
RECORD CONTAINS 80 CHARACTERS
BLOCK CONTAINS 0 RECORDS
DATA RECORD IS MARKIV-RECORD.
01 MARKIV-RECORD.
   02 FILLER                               PIC X(80) .
WORKING-STORAGE SECTION.
01 FILLER                                  PIC X(15) VALUE 'WORKING STORAGE' .
LINKAGE SECTION.
01 MARKIV-PARAMETER-AREA SYNC.
   02 MARKIV-FUNCTION                      PIC S9(8) COMP.
      88 OPEN-FILE                        VALUE +4.
      88 CLOSE-FILE                       VALUE +8.
      88 GET-SEQUENTIAL                    VALUE +12.
      88 ADD-SEQUENTIAL                    VALUE +16.
      88 DELETE-SEQ-UIP                   VALUE +20.
      88 REPLACE-SEQ-UIP                  VALUE +24.
      88 ADD-SEQ-UIP                      VALUE +28.
      88 DELETE-KEY-UIP                   VALUE +32.
      88 REPLACE-KEY-UIP                  VALUE +36.
      88 ADD-KEY-UIP                      VALUE +40.
      88 GET-KEY-UIP                      VALUE +44.
   02 MARKIV-STATUS-CODE                   PIC S9(8) COMP.
      88 STATUS-OK                        VALUE ZEROES.
      88 STATUS-EOF                       VALUE +4.
      88 RECORD-NOT-FOUND                 VALUE +8.
      88 ALLOCATE-WORK                    VALUE +12.
   02 TEMP-STORAGE-START                   PIC S9(8) COMP.
   02 TEMP-STORAGE-LENGTH                 PIC S9(8) COMP.
   02 FILLER                               PIC X(4) .
   02 MARKIV-INPUT-SIZE                    PIC S9(8) COMP.
   02 MARKIV-OUTPUT-SIZE                  PIC S9(8) COMP.
01 MARKIV-FILE-TABLE SYNC.
   02 MARKIV-FILE-NAME                    PIC X(8) .
   02 MARKIV-DDNAME                       PIC X(8) .
   02 MARKIV-FILE-CODE                    PIC S9(4) COMP.
   02 MARKIV-RECORD-FORMAT                PIC S9(4) COMP.
   02 MARKIV-KEY-COUNT                    PIC S9(4) COMP.
   02 MARKIV-RECORD-BLOCKING              PIC S9(4) COMP.
      88 UNBLOCKED-FILE                   VALUE +4.
      88 BLOCKED-FILE                     VALUE +8.
   02 MARKIV-RECORD-SIZE                  PIC S9(8) COMP.
   02 MARKIV-BLOCK-SIZE                   PIC S9(8) COMP.
   02 FILLER                               PIC S9(8) COMP.
   02 MARKIV-ACCESS-MODE                  PIC S9(4) COMP.
      88 RANDOM-ACCESS                     VALUE +4.
      88 SEQUENTIAL-ACCESS                 VALUE +8.
   02 MARKIV-PROCESS-MODE                 PIC S9(4) COMP.
      88 OUTPUT-ONLY                       VALUE +4.
      88 INPUT-ONLY                       VALUE +8.
      88 UPDT-IN-PLACE                     VALUE +12.
   02 MARKIV-START-SEARCH                 PIC X(16) .
   02 MARKIV-END-SEARCH                   PIC X(16) .
   02 MARKIV-SAVED                        PIC X(12) .
   02 MARKIV-WORK-SPACE                   PIC X(16) .
01-MARKIV-REC.
   02 FILLER                               PIC X(80) .
PROCEDURE DIVISION USING
MARKIV-PARAMETER AREA
MARKIV-FILE-TABLE
MARKIV-REC.

IF OPEN-FILE GO TO OPEN-IT-UP.
IF CLOSE-FILE GO TO CLOSE-IT-UP.
IF GET-SEQUENTIAL GO TO READ-IT.
EXHIBIT NAMED 'INVALID' MARKIV-STATUS-CODE.
MOVE -1 TO MARKIV-STATUS-CODE GOBACK.
OPEN-IT-UP.
OPEN INPUT MARKIV-FILE-IN.
MOVE +12 TO MARKIV-STATUS-CODE.
GOBACK.
CLOSE-IT-UP.
CLOSE MARKIV-FILE-IN.

```

Figure 5-10 COBOL User I/O (Page 2 of 3)

```
      MOVE ZEROES TO MARKIV-STATUS-CODE.  
      GOBACK.  
READ-IT.  
  READ MARKIV-FILE-IN  
  AT END MOVE +4 TO MARKIV-STATUS-CODE  
  GOBACK.  
  MOVE ZEROES TO MARKIV-STATUS-CODE.  
  MOVE MARKIV-RECORD TO MARKIV-REC.  
  GOBACK.
```

Figure 5-10 COBOL User I/O (Page 3 of 3)

Checkpoint/Restart provides a variety of methods for checkpointing VISION:Builder runs so that they can be restarted in case of a computer malfunction or operator error, without excessive loss of processing time. Checkpoints are usually specified on lengthy jobs to avoid rerunning the entire job if it should end abnormally. Checkpoints can be written during processing runs.

VISION:Builder uses the MARKDLI entry module for taking basic checkpoints under IMS/ESA and it uses the MARKDLIX entry module for taking extended checkpoints under IMS/ESA. Restart is only permitted with extended checkpoints (that is, entry using MARKDLIX for both checkpoint and restart).

At each checkpoint specified by you, the contents of storage and the state of the peripheral units are saved. Pertinent identifying information is printed at each checkpoint to aid you in restart procedures. When the run is restarted, the peripheral units are reset to the required condition and the saved memory dump is loaded into storage. Restart at a selected checkpoint is initiated using standard operating system procedures (OS/390 or IMS/ESA) and is transparent to VISION:Builder.

VISION:Builder can write checkpoints automatically at selected intervals during a run or can write them under operator control. In addition, a combination of checkpoint options can be selected to ensure that checkpoints are taken at the appropriate intervals. For example, a checkpoint can be requested under operator control every 30 minutes. If the operator fails to do so within a 30 minute interval, VISION:Builder initiates a checkpoint automatically.

Checkpoint Options

All checkpoint options are specified on the checkpoint specification (CP) statement of the RC group. The CP statement is entered after the RC statement and before the first request (if any). Checkpoints are written on a separate checkpoint tape or direct-access file and are also displayed on the JES log.

The checkpoint COUNT record interval specifies the number of records processed before the checkpoint is taken, but only in the case of read-only or newly inserted records.

For existing records being updated or deleted, both record actions (get and replace or delete) are considered in the counting process and as such, two internal counts are taken for one record. The count is reached at an earlier point and the checkpoint is issued when approximately half the records are processed.

For example, suppose you set the checkpoint count at 6. In action A, six new records are inserted and, in action B, three existing records are deleted. Given these circumstances, the checkpoints are taken as follows:

A — Insert 6 new records	Internal count
Insert record 1	1
Insert record 2	2
Insert record 3	3
Insert record 4	4
Insert record 5	5
Insert record 6	6 (matches specified count)
<hr/>	
6 records processed	CHECKPOINT TAKEN

B — Delete 3 old records	Internal count
Read record 1	1
Delete record 1	2
Read record 2	3
Delete record 2	4
Read record 3	5
Delete record 3	6 (matches specified count)
<hr/>	
3 records processed	CHECKPOINT TAKEN

The following IMS/ESA checkpoint options can be selected:

Checkpoint Files

Checkpoints can be written on a single checkpoint file or on two alternating checkpoint files.

Sort Program Checkpoints

Sort control statements can be created by VISION:Builder that cause the sort program to take checkpoints. The additional parameter is added to the sort control statements when requested on the checkpoint specification statement.

Checkpoints Under Timer Control

VISION:Builder can take a checkpoint at an elapsed time interval. The time interval used is based on the time made available by the operating system.

Checkpoints Under Record Count Control

Checkpoints can be taken based on a count of I/O records on a specified file. This option is useful in a multi-programming environment when elapsed time is not always a true indication of processing time.

Checkpoints Under Operator Control

The operator can control when a checkpoint is taken using the console. This option allows checkpoints to be taken at critical points.

See [IMS/ESA Checkpoint/Restart on page 6-8](#) for IMS/ESA checkpoint options.

Taking Checkpoints in VISION:Builder

This section describes how OS/390 checkpoints are written in VISION:Builder.

Checkpoint Files

In general, checkpoints can be written in two ways:

- On a single checkpoint file.
- On alternating checkpoint files.

Checkpoints on single checkpoint files are written sequentially. When no more space is available on the checkpoint file, standard operating system procedures for multiple checkpoint volumes are used. Checkpoints can also be taken on two alternating files. In this way, only the last two checkpoints are maintained. Each checkpoint is written over the oldest one, so that the latest valid checkpoint is still available if a malfunction occurs while taking the checkpoint.

With alternating checkpoints, two direct-access files are usually allocated with enough space in each file for one checkpoint. Alternating checkpoints can be written on tape; however, this is an inefficient use of physical devices, since only one checkpoint can be written on each tape. The first checkpoint is written on the file specified in Checkpoint File on the CP statement. The second is written on the file specified in Alternating Checkpoint File, the third checkpoint is written over the first, and so on.

Checkpointing at Time Intervals

Checkpoints can be written at intervals from 1 to 99 seconds or 1 to 999 minutes, reflecting the wall clock or program CPU time used by VISION:Builder (depending on the operating system). The time interval begins after request decoding or at the beginning of report generation. If the hardware timer is not on, the operator is informed and can either turn it on or terminate the run.

Checkpointing on Record Count

Checkpoints can be controlled by a count of I/O records processed. The number of records processed between checkpoints can be 1 to 999999. The files which can control checkpoints are shown in the following list. Normally, the old master file would be used during a processing run and the M4REPI file during a report generation run.

M4LIST
M4REPI
M4REPO
M4REPn
M4OLD
M4NEW⁴
M4TRAN⁴
M4AUDIT⁴
M4REJECT⁴
M4CORDn
M4SUBFn

Note: The M4OLD file is the only valid file for use with IMS/ESA checkpoint calls.

Checkpointing Under Operator Control

The operator can request checkpoints during processing by interrupting the program using the console and responding to a VISION:Builder message. Checkpoints cannot be requested by the operator during VISION:Builder source statement decoding.

Checkpointing at End of Volume

Checkpoints can be taken when an end of volume condition occurs on all files or on one specified file, for example, M4OLD, M4REPO, or M4TRAN⁴. When end of volume occurs, the checkpoint is written after processing the current master file record or at the beginning of the next physical page during report generation.

Multiple Checkpoint Options

A checkpoint is always written at the operator's request or when end of volume occurs on a file specified in end of volume on file on the CP statement. Either of these checkpoints automatically resets the time and record intervals if they are also selected for controlling checkpoints. A time interval-initiated checkpoint resets the record count interval and vice versa. If both time and record intervals are selected, the time between two checkpoints is the specified time interval or the time required to process the specified number of records, whichever comes first.

Writing Checkpoints

During the processing phase, checkpoints are written after all processing of a particular master file record is completed. Checkpoints are always taken at this location regardless of the method used to activate the checkpoint.

When a checkpoint is written during processing runs, VISION:Builder prints the following information on M4LIST:

- Checkpoint identification number (from the operating system).
- File containing checkpoint data (in case of alternating option).
- Time of the checkpoint.
- Reason for the checkpoint (such as timer or operator).
- Keys of the master file record just processed.

OS/390 Samples of Checkpoint/Restart

The OS/390 CHKPT macro is used to write checkpoints on sequential or partitioned data sets. VISION:Builder uses the checkpoint identification generated by the operating system, which consists of the letter C followed by a seven digit decimal number. The number begins at C0000001 and ends with the number of checkpoints taken in the job.

You can specify data set organization and block size on the checkpoint file DD statement. The only required DD statement entries are the DSNNAME and DISP parameters. If the data set is on a direct-access device, a space allocation is also required if DISP=NEW. VISION:Builder assumes DSORG=PS if a data set organization is not specified and DISP=NEW.

When deferred restart is used, the only DD statements required are for the files opened at the time checkpoints are written. Since the M4LIB and M4INPUT data sets are closed before checkpoints are written during processing runs, their DD statements can be omitted when the job is restarted. Similarly, the M4INPUT DD statement is not required when restarting a report generation run.

During the report generation step, checkpoints are written immediately before beginning a new page. The operating system outputs checkpoint identification information on the operator's console during both processing and report generation runs.

This section contains two examples of checkpointing and restarting VISION:Builder jobs under OS/390. The first example shows a record selection run and the second shows a report generation run.

In the first example, a report file is generated from an old master file. Checkpoints are being written at 30 minute intervals on a single DASD checkpoint file.

[Figure 6-1](#) shows the run setup for the processing and [Figure 6-2](#) shows the setup

necessary to restart the job from the second checkpoint, taken 60 minutes after processing began. Presumably, the run terminated abnormally after the second checkpoint.

```
//checkpt JOB (accounting information)
//JOB LIB DD DSN=your.builder.loadlib,DISP=SHR,
//step1 EXEC PGM=MARKIV,REGION=1536K
//M4LIST DD SYSOUT=a
//M4LIB DD DSN=your.m4lib,DISP=SHR
//M4REPO DD DSN=your.m4repo,UNIT=sysda,
// SPACE=(TRK,(nn,nn),RLSE),
// DISP=(NEW,KEEP)
//M4SORT DD DSN=sort.file,UNIT=SYSDA,SPACE=(TRK,(nn,nn),RLSE),
// DISP=(NEW,KEEP)
//M4OLD DD DSN=old.master.file,DISP=SHR
//M4CHK1 DD DSN=chkpoint.file,DISP=(,DELETE,CATLG),UNIT=SYSDA,
// SPACE=(TRK,(nn,nn),RLSE)
//M4INPUT DD *
CONTROL
FILE MASTER INPUT, NAME . . .
FILE REPORT
CHECKPOINT TIME 30 MINUTES
.
.
/* VISION:Builder SOURCE STATEMENTS
//
```

Figure 6-1 Checkpointing on an OS/390 Processing Run

Notes

```
1 //checkpt JOB (accounting information),RESTART=(step1,C0000002)
//JOB LIB DD DSN=your.builder.loadlib,DISP=SHR
2 //SYSCHK DD DSN=chkpoint.file,DISP=SHR
//step1 EXEC PGM=MARKIV,REGION=1536K
//M4LIST DD SYSOUT=a
3 //M4REPO DD DSN=your.m4repo,DISP=MOD
4 //M4SORT DD DD=sort.file,DISP=MOD
//M4OLD DD DSN=old.master.file,DISP=SHR
5 //M4CHK1 DD DSN=rstchkpt.file,DISP=(,DELETE,CATLG),UNIT=SYSDA,
// SPACE=(TRK,(nn,nn),RLSE)
//
```

Figure 6-2 Deferred Restart of an OS/390 Processing Run

The following explanations are keyed to the numbered statements in [Figure 6-2](#).

Note: The M4LIB and M4INPUT DD statements need not be included because they are closed after request decoding and before processing of the master file.

Notes	Explanation
1	The RESTART parameter specifies that job CHECKPT is to be restarted from step1 at checkpoint C000002 on the data set with a ddname of SYSCHK.
2	The SYSCHK DD statement contains the DSN of the M4CHK1 DD from the checkpoint run being restarted. It must be placed before the first EXEC statement.
3	M4REPO is changed to a disposition of MOD to accommodate the restart.
4	M4SORT is changed to a disposition of MOD to accommodate the restart.
5	This is a new checkpoint file DSN for logging checkpoints subsequent to the restart.

The following example shows checkpoints being written on two alternating checkpoint files during a report step. Checkpoints are written at intervals of 10,000 M4REPI records. [Figure 6-3](#) and [Figure 6-4](#) show the run setups for report generation and restart. The run is restarted after processing 40,000 report file records.

Notes

```

//checkpt JOB (accounting information)
//JOBLIB DD DSN=your.builder.loadlib,DISP=SHR,
//step1 EXEC PGM=MARKIV,REGION=1536K
//M4LIST DD SYSOUT=a
1 //M4REPI DD DSN=your.m4repo,DISP=OLD
2 //M4CHK1 DD DSN=chkpoint.file.a,DISP=(,DELETE,CATLG),UNIT=SYSDA,
// SPACE=(TRK,(nn,nn),RLSE)
3 //M4CHK2 DD DSN=chkpoint.file.b,DISP=(,DELETE,CATLG),UNIT=SYSDA,
// SPACE=(TRK,(nn,nn),RLSE)
//M4INPUT DD *
CONTROL
FILE REPORT
4 CHECKPOINT FILE M4REPI, COUNT 10000, ALTERNATING
.
.
.
/*
//

```

Figure 6-3 Checkpointing a Report Generation Run

The following explanations are keyed to the numbered statements in [Figure 6-3](#).

Notes	Explanation
1	This report file must already exist and be cataloged.
2	The odd checkpoints will be written on M4CHK1.
3	The even checkpoints will be written on M4CHK2.
4	The CP statement specifies that alternating checkpoints are to be written on the data sets M4CHK1 and M4CHK2 at intervals of 10,000 M4REPI records.

Notes

```

1 //checkpt JOB (accounting information),RESTART=(step1,C0000004)
//JOBLIB DD DSN=your.builder.loadlib,DISP=SHR,
2 //SYSCHK DD DSN=chkpoint.file.b,DISP=SHR
//step1 EXEC PGM=MARKIV,REGION=1536K
//M4LIST DD SYSOUT=a
//M4REPI DD DSN=your.m4repo,DISP=OLD
3 //M4CHK1 DD DSN=rstchkpt.file.a,DISP=(,DELETE,CATLG),UNIT=SYSDA,
// SPACE=(TRK,(nn,nn),RLSE)
3 //M4CHK2 DD DSN=rstchkpt.file.b,DISP=(,DELETE,CATLG),UNIT=SYSDA,
// SPACE=(TRK,(nn,nn),RLSE)

```

Figure 6-4 Deferred Restart of a Report Generation Run

The following explanations are keyed to the numbered statements in [Figure 6-4](#).

Notes	Explanation
1	The RESTART parameter specifies that job CHECKPT is to be restarted from step1 at checkpoint C0000004 on the data set with the SYSCHK ddname.
2	The SYSCHK DD statement contains the DSN of the M4CHK1 or M4CHK2 DD (depending on which checkpoint is used for the restart) from the checkpoint run. It must be placed before the first EXEC statement.
3	These are new checkpoint file DSNs for logging checkpoints subsequent to the restart.

IMS/ESA Checkpoint/Restart

Checkpoints can be taken by VISION:Builder running in either a batch region or a batch message processing (BMP) region. The CP statement is used to specify that checkpoints are to be taken and the frequency at which they are to occur.

IMS/ESA Requirements for Checkpointing

When you want checkpoints with the potential for restart, the entry point name to VISION:Builder must be MARKDLIX (that is, MARKDLIX must be specified as the application program name parameter in IMS/ESA region controller JCL). When the MARKDLIX entry point is used, the GSAM access method is used for sequential files (except M4INPUT and M4SORT) whenever the QSAM access method would ordinarily have been used. This requires that a GSAM DBD be provided for each file which utilizes the GSAM access method. In addition, the PSB must include a PCB for each GSAM file. The PSB must also specify CMPAT=YES on the PSBGEN statement.

For each GSAM DBD, the DBD name must be identical to the file name parameter on the corresponding RF statement for the file. If no RF statement is present (as will be the case for the M4LIST file), the GSAM DBD name must be identical to the DD name for the file (M4LIST for the source listing file, M4TRAN for the transaction file, and so on). The DD name in your JCL must correspond to the DD1 parameter in your GSAM DBD.

The file characteristic parameters (RECFM, RECORD, and SIZE) on the DATASET statement of the DBD must be consistent with the data set and the VISION:Builder file definition or transaction definition if an FD or TD applies to the file. Otherwise, an unexpected PCB status code, erroneous input data, or loss of output data may occur.

[Figure 6-5](#) shows sample GSAM DBDs and [Figure 6-6](#) shows a sample PSB including GSAM PCBs.

```

DBD NAME=M4LIST,ACCESS=(GSAM,BSAM)
DATASET DD1=M4LIST,RECFM=FB,RECORD=133,SIZE=2660

DBDGEN FINISH
END

DBD NAME=M4REPO,ACCESS=(GSAM,BSAM)
DATASET DD1=M4REPO,RECFM=VB,RECORD=1020,SIZE=1024
DBDGEN
FINISH
END

DBD NAME=M4TRAN,ACCESS=(GSAM,BSAM)
DATASET DD1=M4TRAN,RECFM=FB,RECORD=80,SIZE=800
DBDGEN
FINISH
END

DBD NAME=M4CORD1,ACCESS=(GSAM,BSAM)
DATASET DD1=M4CORD1,RECFM=FB,RECORD=80,SIZE=2000
DBDGEN
FINISH
END

```

Figure 6-5 Sample GSAM DBDs

```

PCB TYPE=DB, DBDNAME=EMPXDBD, PROCOPT=AP, KEYLEN=50, POS=M
SENSEG NAME=EMPLOYEE, PARENT=0
SENSEG NAME=HISTORY, PARENT=EMPLOYEE
SENSEG NAME=PROJHIST, PARENT=HISTORY
SENSEG NAME=PROJACCT, PARENT=PROJHIST
SENSEG NAME=SKILHIST, PARENT=HISTORY
SENSEG NAME=ADMIN, PARENT=EMPLOYEE
SENSEG NAME=PERSONAL, PARENT=EMPLOYEE
SENSEG NAME=CARS, PARENT=PERSONAL
SENSEG NAME=SCHOOLS, PARENT=PERSONAL
SENSEG NAME=DEGREES, PARENT=SCHOOLS
SENSEG NAME=SUBJECTS, PARENT=DEGREES
SENSEG NAME=WORK, PARENT=EMPLOYEE
SENSEG NAME=EXTS, PARENT=WORK
*
PCB TYPE=GSAM, DBDNAME=M4LIST, PROCOPT=L
*
PCB TYPE=GSAM, DBDNAME=M4REPO, PROCOPT=L
*
PCB TYPE=GSAM, DBDNAME=M4TRAN, PROCOPT=G
*
PCB TYPE=GSAM, DBDNAME=M4CORD1, PROCOPT=G
*
PSBGEN PSBNAME=EMPPSB, LANG=ASSEM, CMPAT=YES
END

```

Figure 6-6 Sample PSB with GSAM

To initiate a restart from a previous batch IMS/ESA checkpoint run, supply a value for the CKPTID parameter on the EXEC statement of the restart JCL with the value in message DFS0540I on the JES JOB LOG. That value is the concatenation of the 2-digit REGID, 3-digit DAY, and 7-digit TIME. If the restart is from a BMP checkpoint run, use either the same concatenation, the value of the CKPTID keyword (also found in message DFS0540I) or the four-character value LAST. [Figure 6-7](#) and [Figure 6-8](#) show sample JCL for checkpoint and restart jobs requested in a batch region. [Figure 6-9 on page 6-12](#) and [Figure 6-10 on page 6-13](#) show sample JCL for checkpoint and restart jobs requested in a BMP region.

```

//imsckpt JOB (accounting information)
//*
//IMSDLI PROC MBR=TEMPNAME, PSB=, BUF=7,
//          SPIE=0, TEST=0, EXCPVR=0, RST=0, PRLD=,
//          SRCH=0, CKPTID=, MON=N, LOGA=0, FMTO=T,
//          IMSID=, SWAP=, DBRC=, IRLM=, IRLMNM=, BKO=N
//GO EXEC PGM=DFSRRCO0,
//        PARM=(DLI, &MBR, &PSB, &BUF,
//              &SPIE&TEST&EXCPVR&RST, &PRLD,
//              &SRCH, &CKPTID, &MON, &LOGA, &FMTO,
//              &IMSID, &SWAP, &DBRC, &IRLM, &IRLMNM, &BKO)
//        PEND
//
//M4P EXEC IMSDLI, MBR=MARKDLIX, PSB=yourpsb, REGION=1500K
//STEPLIB DD DSN=IMSVS.RESLIB, DISP=SHR
//        DD DSN=your.builder.loadlib, DISP=SHR
//        DD DSN=your.comlib.loadlib, DISP=SHR
//DFSRESLB DD DSN=IMSVS.RESLIB, DISP=SHR
//IMS DD DSN=your.psblib, DISP=SHR
//        DD DSN=your.dbdlib, DISP=SHR
//IEFRDER DD DSN=your.ckp1.imslog, DISP=(NEW, CATLG),
//        UNIT=ISPDA, SPACE=(TRK, (5, 2), RLSE),
//        DCB=(RECFM=VB, BLKSIZE=1920, LRECL=1916, BUFNO=2)
//M4LIB DD DSN=your.m4lib, DISP=SHR
//M4LIST DD DSN=your.gsam.m4list, DISP=(NEW, CATLG),
//        UNIT=ISPDA, SPACE=(TRK, (10, 5), RLSE)

```

Figure 6-7 Checkpointing on an IMS/ESA Processing Run (Page 1 of 2)

```

//M4REPO DD DSN=your.gsam.m4repo,DISP=(NEW,CATLG),
//
//M4SORT DD UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(TRK,1)
//M4INPUT DD DSN=your.builder.program,DISP=SHR
//
// DSN=your.ims.database,DISP=SHR
//
//GENER EXEC PGM=IEBGENER,COND=EVEN
//SYSPRINT DD DUMMY
//SYSUT1 DD DSN=*.M4P.GO.M4LIST,DISP=OLD
//SYSUT2 DD SYSOUT=A,DCB=(RECFM=FA,LRECL=133,BLKSIZE=133)
//SYSIN DD DUMMY
//
//SORT EXEC PGM=SORT,REGION=1500K,COND=(0,NE,M4P.GO)
//SYSOUT DD SYSOUT=A
//SORTIN DD DSN=*.M4P.GO.M4REPO,DISP=OLD
//SORTOUT DD UNIT=SYSDA,DISP=(NEW,PASS),
//
// SPACE=(TRK,(10,5),RLSE)
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN DD DSN=*.M4P.GO.M4SORT,DISP=(OLD,DELETE)
//
//M4R EXEC PGM=MARKIV,REGION=1536K,COND=(0,NE,SORT)
//STEPLIB DD DSN=your.builder.loadlib,DISP=SHR
//
// DSN=your.comlib.loadlib,DISP=SHR
//M4LIST DD SYSOUT=A
//M4REPI DD DSN=*.SORT.SORTOUT,DISP=(OLD,DELETE)
//M4INPUT DD DSN=your.builder.run.control,DISP=SHR
//

```

Figure 6-7 Checkpointing on an IMS/ESA Processing Run (Page 2 of 2)

```

//imsckpt JOB (accounting information)
//
//IMSDLI PROC MBR=TEMPNAME,PSB=,BUF=7,
//
// SPIE=0,TEST=0,EXCPVR=0,RST=0,PRLD=,
//
// SRCH=0,CKPTID=,MON=N,LOGA=0,FMTO=T,0
//
// IMSID=,SWAP=,DBRC=,IRLM=,IRLMNM=,BKO=N
//GO EXEC PGM=DFSRR00,
//
// PARM=(DLI,&MBR,&PSB,&BUF,
//
// &SPIE&TEST&EXCPVR&RST,&PRLD,
//
// &SRCH,&CKPTID,&MON,&LOGA,&FMTO,
//
// &IMSID,&SWAP,&DBRC,&IRLM,&IRLMNM,&BKO)
//
// PEND
//
//M4P EXEC IMSDLI,MBR=MARKDLIX,PSB=yourpsb,CKPTID=nnnnnnnnnnnn,
//
// REGION=1500K
//STEPLIB DD DSN=IMSVS.RESLIB,DISP=SHR
//
// DSN=your.builder.loadlib,DISP=SHR
//
// DSN=your.comlib.loadlib,DISP=SHR
//DFSRESLB DD DSN=IMSVS.RESLIB,DISP=SHR
//IMS DD DSN=your.psbllib,DIPS=SHR
//
// DSN=your.dbdlib,DISP=SHR
//IMSLOGR DD DSN=your.ckp1.imslog,DIPS=OLD
//IEFRDER DD DSN=your.ckp2.imslog,DISP=(NEW,CATLG),
//
// UNIT=ISPDA,SPACE=(TRK,(5,2),RLSE),
//
// DCB=(RECFM=VB,BLKSIZE=1920,LRECL=1916,BUFNO=2)
//M4LIB DD DSN=your.m4lib,DISP=SHR
//M4LIST DD DSN=your.gsam.m4list,DISP=OLD
//M4REPO DD DSN=your.gsam.m4repo,DISP=OLD
//M4SORT DD UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(TRK,1)
//M4INPUT DD DSN=your.builder.source,DISP=SHR
//... DD DSN=your.ims.database,DISP=SHR
//
//GENER EXEC PGM=IEBGENER,COND=EVEN
//SYSPRINT DD DUMMY
//SYSUT1 DD DSN=*.M4P.GO.M4LIST,DISP=OLD
//SYSUT2 DD SYSOUT=A,DCB=(RECFM=FA,LRECL=133,BLKSIZE=133)
//SYSIN DD DUMMY
//
//SORT EXEC PGM=SORT,REGION=1500K,COND=(0,NE,M4P.GO)
//SYSOUT DD SYSOUT=A

```

Figure 6-8 Restarting an IMS/ESA Processing Run (Page 1 of 2)

```

//SORTIN DD DSN=* .M4P.GO.M4REPO,DISP=OLD
//SORTOUT DD UNIT=SYSDA,DISP=(NEW,PASS),
//          SPACE=(TRK,(10,5),RLSE)
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN DD DSN=* .M4P.GO.M4SORT,DISP=(OLD,DELETE)
//*
//M4R EXEC PGM=MARKIV,REGION=1536K,COND=(0,NE,SORT)
//STEPLIB DD DSN=your.builder.loadlib,DISP=SHR
//          DD DSN=your.comlib.loadlib,DISP=SHR
//M4LIST DD SYSOUT=A
//M4REPI DD DSN=* .SORT.SORTOUT,DISP=(OLD,DELETE)
//M4INPUT DD DSN=your.builder.run.control,DISP=SHR
//

```

Figure 6-8 Restarting an IMS/ESA Processing Run (Page 2 of 2)

```

//bmprun JOB (accounting information)
//*
//IMSBMP PROC MBR=TEMPNAME,PSB=,IN=,OUT=,
//          OPT=N,SPIE=0,TEST=0,DIRCA=000,PRLD=,STIMER=,
//          CKPTID=,PARDLI=,CPUTIME=,NBA=,OBA=,IMSID=,AGN=
//BMP EXEC PGM=DFSRRRC00,
//          PARM=(BMP,&MBR,&PSB,&IN,&OUT,
//          &OPT&SPIE&TEST&DIRCA,&PRLD,&STIMER,&CKPTID,
//          &PARDLI,&CPUTIME,&NBA,&OBA,&IMSID,&AGN)
//          PEND
//*
//M4P EXEC IMSBMP,MBR=MARKDLIX,PSB=yourpsb,REGION=1536K
//STEPLIB DD DSN=IMSVS.RESLIB,DISP=SHR
//          DD DSN=IMSVS.PGMLIB,DISP=SHR
//          DD DSN=your.builder.loadlib,DISP=SHR
//          DD DSN=your.comlib.loadlib,DISP=SHR
//IMS DD DSN=IMSVS.DBDLIB,DISP=SHR
//          DD DSN=IMSVS.PSBLIB,DISP=SHR
//M4LIB DD DSN=your.m4lib,DISP=SHR
//M4TRAN DD DSN=your.gsam.m4tran,DISP=SHR
//M4REPO DD DSN=your.gsam.m4repo,DISP=(NEW,CATLG),
//          UNIT=ISPD,SPACE=(TRK,(10,5),RLSE)
//M4LIST DD DSN=your.gsam.m4list,DISP=(NEW,CATLG),
//          UNIT=ISPD,SPACE=(TRK,(10,5),RLSE)
//M4SORT DD UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(TRK,1)
//M4INPUT DD DSN=your.builder.source,DISP=SHR
//*
//GENER EXEC PGM=IEBGENER,COND=EVEN
//SYSPRINT DD DUMMY
//SYSUT1 DD DSN=* .M4P.BMP.M4LIST,DISP=OLD
//SYSUT2 DD SYSOUT=A,DCB=(RECFM=FA,LRECL=133,BLKSIZE=133)
//SYSIN DD DUMMY
//*
//SORT EXEC PGM=SORT,REGION=2000K,COND=EVEN
//SYSOUT DD SYSOUT=A
//SORTIN DD DSN=* .M4P.BMP.M4REPO,DISP=OLD
//SORTOUT DD DSN=&&REPI,UNIT=SYSDA,DISP=(NEW,PASS),
//          SPACE=(TRK,(10,5),RLSE)
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN DD DSN=* .M4P.BMP.M4SORT,DISP=(OLD,DELETE)
//*
//M4R EXEC PGM=MARKIV,REGION=1536K,COND=(0,NE,SORT)
//STEPLIB DD DSN=your.builder.loadlib,DISP=SHR
//          DD DSN=your.comlib.loadlib,DISP=SHR
//M4LIST DD SYSOUT=A
//M4REPI DD DSN=&&REPI,DISP=(OLD,DELETE)
//M4INPUT DD DSN=your.report.run.control,DISP=SHR
//

```

Figure 6-9 Checkpointing on an IMS/ESA BMP Processing Run

```

//bmprun JOB (accounting information)
//*
//IMSBMP PROC MBR=TEMPNAME, PSB=, IN=, OUT=,
// OPT=N, SPIE=0, TEST=0, DIRCA=000, PRLD=, STIMER=,
// CKPTID=, PARDLI=, CPUTIME=, NBA=, OBA=, IMSID=, AGN=
//BMP EXEC PGM=DFSRRRC00,
// PARM=(BMP, &MBR, &PSB, &IN, &OUT,
// &OPT&SPIE&TEST&DIRCA, &PRLD, &STIMER, &CKPTID,
// &PARDLI, &CPUTIME, &NBA, &OBA, &IMSID, &AGN)
// PEND
//*
//M4P EXEC IMSBMP, MBR=MARKDLIX, PSB=yourpsb, REGION=1536K, CKPTID=LAST
//STEPLIB DD DSN=IMSVS.RESLIB, DISP=SHR
// DD DSN=IMSVS.PGMLIB, DISP=SHR
// DD DSN=your.builder.loadlib, DISP=SHR
// DD DSN=your.comlib.loadlib, DISP=SHR
//IMS DD DSN=IMSVS.DBDLIB, DISP=SHR
// DD DSN=IMSVS.PSBLIB, DISP=SHR
//M4LIB DD DSN=your.m4lib, DISP=SHR
//M4TRAN DD DSN=your.gsam.m4tran, DISP=SHR
//M4REPO DD DSN=your.gsam.m4repo, DISP=OLD
//M4LIST DD DSN=your.gsam.m4list, DISP=OLD
//M4SORT DD UNIT=SYSDA, DISP=(NEW, PASS), SPACE=(TRK, 1)
//M4INPUT DD DSN=your.builder.source, DISP=SHR
//*
//GENER EXEC PGM=IEBGENER, COND=EVEN
//SYSPRINT DD DUMMY
//SYSUT1 DD DSN=*.M4P.BMP.M4LIST, DISP=OLD
//SYSUT2 DD SYSOUT=A, DCB=(RECFM=FA, LRECL=133, BLKSIZE=133)
//SYSIN DD DUMMY
//*
//SORT EXEC PGM=SORT, REGION=2000K, COND=EVEN
//SYSOUT DD SYSOUT=A
//SORTIN DD DSN=*.M4P.BMP.M4REPO, DISP=OLD
//SORTOUT DD DSN=&&REPI, UNIT=SYSDA, DISP=(NEW, PASS),
// SPACE=(TRK, (10, 5), RLSE)
//SORTWK01 DD UNIT=SYSDA, SPACE=(CYL, (1, 1))
//SORTWK02 DD UNIT=SYSDA, SPACE=(CYL, (1, 1))
//SORTWK03 DD UNIT=SYSDA, SPACE=(CYL, (1, 1))
//SYSIN DD DSN=*.M4P.BMP.M4SORT, DISP=(OLD, DELETE)
//*
//M4R EXEC PGM=MARKIV, REGION=1536K, COND=(0, NE, SORT)
//STEPLIB DD DSN=your.builder.loadlib, DISP=SHR
// DD DSN=your.comlib.loadlib, DISP=SHR
//M4LIST DD SYSOUT=A
//M4REPI DD DSN=&&REPI, DISP=(OLD, DELETE)
//M4INPUT DD DSN=your.report.run.control, DISP=SHR
//

```

Figure 6-10 Restarting an IMS/ESA BMP Processing Run

VISION:Builder Considerations for IMS/ESA Checkpoints

Checkpoint frequency on the CP statement can be specified either as a time interval or as a record count interval for M4OLD.

When specifying M4REPO in the JCL for Checkpoint/Restart runs, it is necessary to retain the M4REPO data set so that it is available should a RESTART run be necessary. If a temporary data set is used and subsequently deleted, sort problems can occur and no report step is generated in the restart run.

Three flag fields, CHKP, RESTART, and CKPTID, are provided to facilitate user applications utilizing checkpoints. The CHKP flag field can be used to trigger a checkpoint based upon user-determined criteria by storing a non-blank value into

the flag. This action triggers a checkpoint operation when the next master file root segment is about to be read. The CHKP flag field is reset to a blank after every checkpoint operation.

The RESTART flag field can be used by requests to determine the restart status of a VISION:Builder run. If a run has been restarted, the RESTART flag contains the checkpoint ID at which the restart occurred. Otherwise, the value of the RESTART flag is all blanks. The RESTART flag is provided for applications utilizing own-code or GDBI (mapping requests) which could require knowledge of the restart status. The CKPTID flag field contains the checkpoint ID of the last checkpoint taken.

VISION:Builder Restrictions for IMS/ESA Checkpoint/Restart

The following restrictions apply to VISION:Builder applications using the IMS/ESA Checkpoint facility:

- Sorting and report output operations must be run as separate job steps. One-step processing if sorting or report output is specified should be avoided.
- VISION:Builder programs using own-code or GDBI (mapping requests) can take checkpoints and try restarts but only at the user's risk. Successful restart might require changes in the user's code or might even be impossible.
- No program data definition changes can be made before a program is restarted. Minor program logic changes can be made but these changes must not create any additional temporary fields (either default or explicitly defined), change the order of any defined fields (temporary fields or fields defined using an FD statement), reference new flag fields, or affect the number of files or the sequence in which they are accessed.

If a data definition change is detected during restart, VISION:Builder terminates. Note, however, that not all illegal program data definition changes are detected and, in this case, run results are unpredictable.

- Any program which creates a new IMS/ESA or VSAM database (using M4NEW or M4SUBFn) is not eligible for Checkpoint/Restart.
- VSAM files can only be used as indexed coordinated files.
- Packed files (record format of P in the file definition) cannot be used by the VISION:Builder program for Checkpoint/Restart.

IMS/ESA Checkpoints with DB2

The IMS/ESA checkpoint facility can be used with VISION:Builder in runs using DB2 files with or without any IMS/ESA files. When DB2 files are used and VISION:Builder is running in an IMS/ESA batch region, the sample JCL shown in [Figure 6-11](#) must be used in place of the sample JCL shown in [Figure 6-7 on page 6-10](#). Likewise, the restart JCL shown in [Figure 6-8 on page 6-11](#) must be changed in a manner consistent with that shown in [Figure 6-11](#) when performing

an IMS/ESA restart while VISION:Builder is running in an IMS/ESA batch region. In the case where VISION:Builder is running in a BMP region, no JCL change is required, regardless of whether or not any DB2 files are used.

```
//imsckpt JOB (accounting information)
//*
//IMSDLI PROC MBR=TEMPNAME,PSB=,BUF=7,
//          SPIE=0,TEST=0,EXCPVR=0,RST=0,PRLD=,
//          SRCH=0,CKPTID=,MON=N,LOGA=0,FMTO=T,
//          IMSID=,SWAP=,DBRC=,IRLM=,IRLMNM=,BKO=N
//GO       EXEC PGM=DFSRRCO0,
//          PARM=(DLI,&MBR,&PSB,&BUF,
//          &SPIE&TEST&EXCPVR&RST,&PRLD,
//          &SRCH,&CKPTID,&MON,&LOGA,&FMTO,
//          &IMSID,&SWAP,&DBRC,&IRLM,&IRLMNM,&BKO)
//          PEND
//*
//M4P     EXEC IMSDLI,MBR=DSNMTV01,PSB=yourpsb,REGION=1500K
//STEPLIB DD DSN=IMSVS.RESLIB,DISP=SHR
//          DD DSN=your.builder.loadlib,DISP=SHR
//          DD DSN=your.comlib.loadlib,DISP=SHR
//          DD DSN=your.db2.loadlib,DISP=SHR
//DFSRESLB DD DSN=IMSVS.RESLIB,DISP=SHR
//IMS      DD DSN=your.psblib,DISP=SHR
//          DD DSN=your.dbdlib,DISP=SHR
//IEFRDER DD DSN=your.chk1.imslog,DISP=(NEW,CATLG),
//          UNIT=SYSDA,SPACE=(TRK,(5,2),RLSE),
//          DCB=(RECFM=VB,BLKSIZE=1920,LRECL=1916,BUFNO=2)
//DDITV02 DD *
ssid,SYS1,DSNMIN10,,R,-,DLIBATCH,yourplan,MARKDLIX
//DDOTV02 DD UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(TRK,(1,1)),
//          DCB=(RECFM=VB,LRECL=4092,BLKSIZE=4096)
//M4LIB   DD DSN=your.m4lib,DISP=SHR
//M4LIST  DD DSN=your.gsam.m4list,DISP=(NEW,CATLG),
//          UNIT=SYSDA,SPACE=(TRK,(10,5),RLSE)
//M4REPO  DD DSN=your.gsam.m4repo,DISP=(NEW,CATLG),
//          UNIT=SYSDA,SPACE=(TRK,(10,5),RLSE)
//M4SORT  DD UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(TRK,1)
//M4INPUT DD DSN=your.builder.program,DISP=SHR
//database DD DSN=your.ims.database,DISP=SHR
//*
//GENER   EXEC PGM=IEBGENER,COND=EVEN
//SYSPRINT DD DUMMY
//SYSUT1  DD DSN=*.M4P.GO.M4LIST,DISP=OLD
//SYSUT2  DD SYSOUT=*,DCB=(RECFM=FA,LRECL=133,BLKSIZE=133)
//SYSIN   DD DUMMY
//*
//SORT    EXEC PGM=SORT,REGION=1500K,COND=(0,NE,M4P.GO)
//SYSOUT  DD SYSOUT=*
//SORTIN  DD DSN=*.M4P.GO.M4REPO,DISP=OLD
//SORTOUT DD UNIT=SYSDA,DISP=(NEW,PASS),
//          SPACE=(TRK,(10,5),RLSE)
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN   DD DSN=*.M4P.GO.M4SORT,DISP=(OLD,DELETE)
//*
//M4R     EXEC PGM=MARKIV,REGION=1536K,COND=(0,NE,M4P.GO)
//STEPLIB DD DSN=your.builder.loadlib,DISP=SHR
//          DD DSN=your.comlib.loadlib,DISP=SHR
//M4LIST  DD SYSOUT=*
//M4REPI  DD DSN=*.SORT.SORTOUT,DISP=(OLD,DELETE)
//M4INPUT DD DSN=your.builder.run.control.stmt,DISP=SHR
//*
//DB2LST EXEC PGM=DFSERA10,COND=EVEN
//STEPLIB DD DSN=IMSVS.RESLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1  DD DSN=*.M4P.GO.DDOTV02,DISP=(OLD,DELETE)
//SYSIN   DD *
CONTROL  CNTL K=000,H=8000
OPTION   PRINT
//
```

Figure 6-11 IMS/ESA Checkpoints with DB2 in a Batch Region

Query Language Input and Output Files

The Query Language accepts input from a statement image file on tape or disk. All files except SYSPRINT must contain 80-byte fixed length unblocked records. SYSPRINT contains 121-byte fixed length unblocked records. If a COPY command is entered, input can also be obtained from the file specified in the command.

Certain Query Language command statements such as USE, LIST, or SORT are automatically saved in the \$SOURCE file. They can also be saved in other files specified in a RETAIN command. The \$SOURCE file is available to you on exit from the Query Language processor (BQL).

The \$QUERY file contains the VISION:Builder source statements generated by the Query Language processor for the request(s), if any.

The \$GLOSS file contains the VISION:Builder specifications required to obtain a glossary, if one was specified.

Both \$QUERY and \$GLOSS files are available to you on exit from the Query Language processor. These files are normally used to execute the actual requests to produce reports or glossaries.

The \$TEMP file is a temporary work file generated during preparation of the \$QUERY file.

Figure 7-1 shows these I/O files of the Query Language and their relationship to the subsequent VISION:Builder steps. For more information on VISION:Builder steps, job steps, and I/O data files, refer to the [VISION:Builder Reference Guide](#).

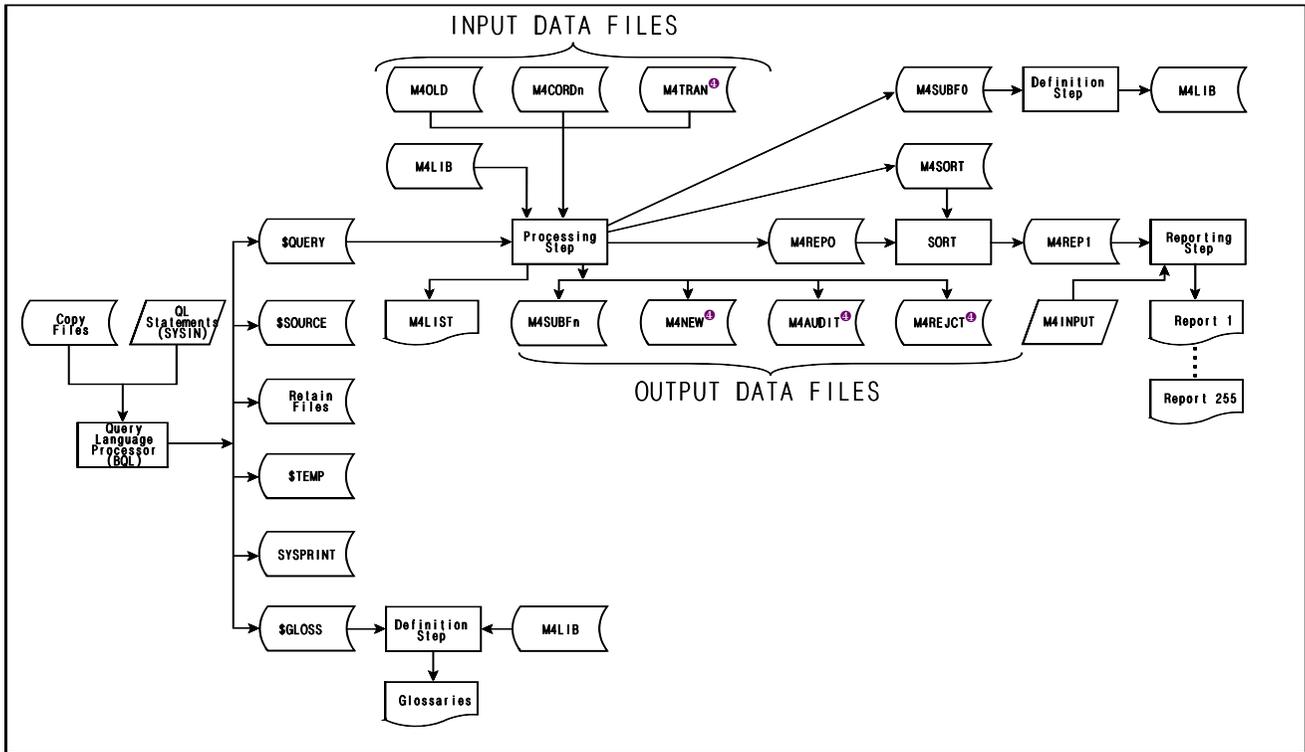


Figure 7-1 Batch Query Language Job-Step Flow and Input/Output Files

Steps in the Execution of the Batch Query Language

This section describes the job steps used for Query Language runs. The QL procedure consists of six steps: QLM, GLS, PRO, DEF, SRT, and REP, as shown in [Figure 7-2 on page 7-3](#). These six job steps make up a complete Query Language job stream. This job stream provides a successful execution of all functional capabilities of the Query Language. Once the job stream is available, you can control the execution based on the kind of query you enter.

The OS/390 versions of the Query Language use operating system return codes to control the execution of the job steps. These job steps are described in the following subsections and the names used correspond to the step names shown in the sample JCL procedure ([Figure 7-2](#)).

Query Step

The query (QLM) step reads the input QL statements and generates up to four output files:

- \$SOURCE contains a copy of the QL input statements.
- \$TEMP is a temporary work file.
- \$QUERY contains the VISION:Builder source statements generated for the request(s), if any. This file is the M4INPUT file in the PRO (processing) step.
- \$GLOSS contains the VISION:Builder source statements needed to obtain a glossary, if one was requested. This file is the M4INPUT file in the GLS (glossary) step.

The QLM step return codes indicate which of the other five Query Language steps in [Figure 7-2](#) are to be executed:

0	PRO
1	GLS, PRO
2	PRO, DEF
3	GLS, PRO, DEF
4	GLS
130	None; a start before END statement or START after query error.
140	None; either there were too many continuation statements, there was no START command, or there was an error in compilation.
150	None; the QUIT command was entered.

Only the above return codes are valid from the QLM step.

A return code of 0 from the execution of the PRO step indicates that the SRT and REP steps should be executed.

A return code of 8 from the execution of the PRO step indicates that the SRT and REP steps are not to be executed.

```
//M4QUERY PROC M4LIB='MARKIV.QUERY.M4LIB',           M4LIB DSNAME
//          OLDMAST=NULLFILE,                         M4OLD DSNAME
//          NEWMAST=NULLFILE,NEWUNIT=,NEWVOL=,        M4NEW INFO
//          TRANS=NULLFILE,CORDONE=NULLFILE,         M4TRAN/M4CORD1 DSN'S
//          SUBONE=NULLFILE,SF1UNIT=,SF1VOL=,        M4SUBF1 INFO
//          SUBTWO=NULLFILE,SF2UNIT=,SF2VOL=        M4SUBF2 INFO
//*
//QLM      EXEC PGM=BQL,REGION=100K
//$QUERY   DD DSN=&&QUERY,UNIT=SYSDA,DISP=(,PASS),
```

Figure 7-2 Sample JCL Procedure for a Query Language Run Under OS/390 (Page 1 of 3)

```

//          DCB=(BLKSIZE=80,LRECL=80,RECFM=F),SPACE=(TRK,(2,1))
//$SOURCE DD DSN=&&SOURCE,UNIT=SYSDA,DISP=(,PASS),
//          DCB=(BLKSIZE=80,LRECL=80,RECFM=F),SPACE=(TRK,(2,1))
//$TEMP  DD UNIT=SYSDA,SPACE=(TRK,(2,1)),
//          DCB=(BLKSIZE=80,LRECL=80,RECFM=F)
//$GLOSS DD DSN=&&GLOSS,UNIT=SYSDA,DISP=(,PASS),
//          DCB=(BLKSIZE=80,LRECL=80,RECFM=F),SPACE=(TRK,1)
//SYSPRINT DD SYSOUT=A
//*
//GLS    EXEC PGM=MARKIV,REGION=1536K,
//          COND=((0,EQ,QLM),(2,EQ,QLM),(4,LT,QLM))
//M4LIB  DD DSN=&M4LIB,DISP=SHR
//M4INPUT DD DSN=*.QLM.$GLOSS,DISP=(OLD,DELETE)
//M4LIST DD SYSOUT=A
//*
//PRO    EXEC PGM=MARKIV,REGION=1536K,COND=(4,LE,QLM)
//M4LIB  DD DSN=&M4LIB,DISP=SHR
//M4INPUT DD DSN=*.QLM.$QUERY,DISP=(OLD,PASS)
//M4OLD  DD DSN=&OLDMAST,DISP=SHR
//M4NEW  DD DSN=&NEWMAST,DISP=(NEW,PASS,DELETE),
//          UNIT=&NEWUNIT,VOL=SER=&NEWVOL,SPACE=(TRK,(5,5))
//M4TRAN DD DSN=&TRANS,DISP=SHR
//M4CORD1 DD DSN=&CORDONE,DISP=SHR
//M4SUBF0 DD DSN=&M4SUBF0,UNIT=SYSDA,DISP=(,PASS),SPACE=(TRK,1)
//M4SUBF1 DD DSN=&SUBONE.DISP=(NEW,PASS.DELETE),
//          UNIT=&SF1UNIT.VOL=SER=&SF1VOL,SPACE=(TRK,(5,5))
//M4SUBF2 DD DSN=&SUBTWO,DISP=(NEW,PASS,DELETE),
//          UNIT=&SF2UNIT,VOL=SER=&SF2VOL,SPACE=(TRK,(5,5))
//M4REPO DD DSN=&REPO,DISP=(NEW,PASS),UNIT=SYSDA,
//          SPACE=(TRK,(40,40))
//M4SORT DD DSN=&M4SORT,UNIT=SYSDA,DISP=(,PASS),SPACE=(TRK,1)
//M4LIST DD SYSOUT=A
//*
//DEF    EXEC PGM=MARKIV,REGION=1536K,
//          COND=((4,EQ,PRO),(8,LT,PRO),(2,GT,QLM),(3,LT,QLM))
//M4LIB  DD DSN=&M4LIB,DISP=SHR
//M4INPUT DD DSN=*.PRO.M4SUBF0,DISP=(OLD,DELETE)
//M4LIST DD SYSOUT=A
//*
//SRT    EXEC PGM=SORT,PARM='CORE=MAX,MSG=AP',REGION=100K,
//          COND=((0,NE,PRO),(4,LE,QLM))
//SORTIN DD DSN=*.PRO.M4REPO,DISP=(OLD,PASS)
//SORTOUT DD DSN=*.PRO.M4REPO,DISP=(OLD,PASS),UNIT=SYSDA,
//          VOL=REF=*.SORTIN
//SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
//SORTWK01 DD SPACE=(CYL,5,,CONTIG),UNIT=(SYSDA,SEP=SORTIN)
//SORTWK02 DD SPACE=(CYL,5,,CONTIG),UNIT=(SYSDA,SEP=SORTWK01)
//SORTWK03 DD SPACE=(CYL,5,,CONTIG),
//          UNIT=(SYSDA,SEP=(SORTWK01,SORTWK02))

```

Figure 7-2 Sample JCL Procedure for a Query Language Run Under OS/390
(Page 2 of 3)

```

//SYSIN    DD DSN=* .PRO.M4SORT,DISP=(OLD,DELETE)
//SYSOUT   DD SYSOUT=A
//*
//REP      EXEC PGM=MARKIV,REGION=1536K,
//          COND=( (0,NE,PRO) , (4,LE,QLM) , (16,LE,SRT) )
//M4INPUT  DD DSN=MARKIV,REPORT.RCCARD,DISP=SHR
//M4LIST   DD SYSOUT=A
//M4REPI   DD DSN=* .SRT.SORTOUT,DISP=(OLD,DELETE)

```

Figure 7-2 Sample JCL Procedure for a Query Language Run Under OS/390
(Page 3 of 3)

Glossary Step

The glossary (GLS) step is a common library definition run to obtain a glossary from the specified library. This step is executed only if the GLOSSARY command was entered in the query language input. The M4INPUT file to this step is the \$GLOSS file generated in the preceding QLM step. Output goes to M4LIST.

The only valid return code for the GLS step is 0. Any other return codes indicate the step failed.

Processing Step

The processing (PRO) step is the VISION:Builder processing step. This step is not executed if only a glossary was requested; otherwise, it uses as M4INPUT the \$QUERY file generated in the previous QLM step. It may read an old master file.

(M4OLD), a transaction file (M4TRAN),⁴ and up to three coordinated files

(M4CORDn). It may output a new master file (M4NEW)⁴ and up to six subfiles (M4SUBF1 through M4SUBF5 and M4SUBF0).

If a file definition is requested for a subfile (M4SUBF1 through M4SUBF5), a M4SUBF0 file is output from this step. This file contains the necessary file definition statements and is the M4INPUT file to the DEF step.

If no sort is required and only one request is specified, the PRO step also generates the required report, which is output to M4LIST. Otherwise, the SRT and REP steps must be executed to do the sort and report.

The default DISP parameter for the VISION:Builder output files (M4NEW,⁴ M4SUBFn) is DISP=(,PASS,DELETE). Note that if these data sets are not kept by a later step, they are passed off the end of the job and deleted. The default DISP

parameter for the VISION:Builder input files (M4OLD, M4TRAN,⁴ M4CORDn) is DISP=SHR.

Valid return codes for the PRO run are 0 and 8. A return code of 0 from the PRO step indicates the SRT and REP steps are required; a return code of 8 indicates no separate SRT/REP steps are required. Any other return codes indicate the step failed.

Definition Step

The definition (DEF) step is a common library definition run. It is executed only if a file definition for M4SUBFn was requested in the Query Language SAVE command. The M4INPUT to this step is the M4SUBF0 generated in the preceding PRO step. Output goes to M4LIST.

If the save library (M4LIB) does not already exist, it must be initialized prior to execution of this step.

The only valid return code for the DEF step is 0. Any other return codes indicate the step failed.

Sort Step

The sort (SRT) step sorts the M4REPO report file output from the PRO step. This step is executed only if you specify a sort or generate more than one VISION:Builder request.

The only valid return code for the SRT step is 0. Any other return codes indicate the step failed.

Report Step

The report (REP) step is a VISION:Builder report step that outputs the required report(s) to M4LIST. This step is executed only if the preceding SRT step was executed.

The only valid return code for the REP step is 0. Any other return codes indicate the step failed.

Sample Execution of Batch Query Language

This section contains a part of the sample JCL used for a batch Query Language run, Query Language input for a report, and the resulting output report. The sample JCL and input are shown in [Figure 7-3](#) and the output report is shown in [Figure 7-4](#).

```
//sample JOB (accounting information),programmer,TIME=(,time),
// REGION=1536K
//*
//JOBLIB DD DSN=MARKIV.DEVEL.AIMSLD,DISP=SHR
//*
//QLRUN EXEC PGM=BQL,REGION=70K
//STEPLIB DD DSN=MARKIV.ONLINE.LOAD,DISP=SHR
//$SOURCE DD DSN=&&SOURCE,UNIT=SYSDA,DISP=(,PASS),SPACE=(TRK,(2,1)),
// DCB=(BLKSIZE=80,LRECL=80,RECFM=F)
//$QUERY DD DSN=&&QUERY,UNIT=SYSDA,DISP=(,PASS),SPACE=(TRK,(2,1)),
// DCB=(BLKSIZE=80,LRECL=80,RECFM=F)
//$TEMP DD DSN=&&TEMP,UNIT=SYSDA,DISP=(,PASS),SPACE=(TRK,(2,1)),
// DCB=(BLKSIZE=80,LRECL=80,RECFM=F)
```

Figure 7-3 Sample JCL Procedure and Batch Query Language Input Statements (Page 1 of 2)

```

//$GLOSS DD DUMMY,DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
      USE OLD LIB
      FNAME CASEPROB
      EQUATE MINEAL='MIN-RAL' PARTNO = 'PART-NO'
      LET T.TOTAL=QANONHND*QANONRD
      LET T.SHORT=MINEAL-T.TOTAL WHERE MINEAL>TOTAL
      LIST PARTNO DESCRIPT QANONHND QANONRD MINEAL T.SHORT
      WHERE MINEAL GT T.TOTAL
      SORT PARTNO
      COUNT PARTNO
      TITLE 'SHORT ITEM REPORT#'
      FORMAT DATEF=A W=B
      END ; START
//MARUN EXEC PGM=MARKIV,COND=(4,LE,QLRUN),REGION=196K
//MALIB DD DSN=DSG.TRAINL.MALIB,DISP=SHR
//MOLD DD DSN=PSOTRAN.EFFARAH.MAIN,DISP=SHR
//MALIST DD SYSOUT=A
//MAREPO DD DSN=@MAREPO,DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(2,1),RLSE)
//MASORT DD DSN=@MASORT,DISP=(,PASS),UNIT=SYSDA,SPACE=(TRK,1)
//MAINPUT DD DSN=@QUERY,DISP=(OLD,DELETE)
//MASORT EXEC PGM=SORT,COND=((4,LE,QLRUN),(4,LE,MARUN)),
//      PARM='MSG=AP,CORE=MAX',REGION=100K
//SORTIN DD DSN=@MAREPO,DISP=(OLD,DELETE)
//SORTOUT DD DSN=@MAREPI,DISP=(,PASS),SPACE=(CYL,(2,1),RLSE)
//      DCB=(RECFM=VB,LRECL=1020,BLKSIZE=1024),UNIT=SYSDA
//SORTWK01 DD SPACE=(CYL,5,,CONTIG),UNIT=(SYSDA,SEP=SORTIN)
//SORTWK02 DD SPACE=(CYL,5,,CONTIG),UNIT=(SYSDA,SEP=SORTWK01)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,5,,CONTIG)
//SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
//SYSOUT DD DUMMY
//SYSIN DD DSN=@MASORT,DISP=(OLD,DELETE)
//MARPT EXEC PGM=MARKIV,REGION=170K,
//      COND=((4,LE,QLRUN),(4,LE,MARUN),(16,LE,MASORT))
//MALIST DD SYSOUT=A
//MAREPI DD DSN=@MAREPI,DISP=(OLD,DELETE,KEEP)
//MAINPUT DD *
MAREPORTRCD S
//

```

(DCB deliberately omitted)

Figure 7-3 Sample JCL Procedure and Batch Query Language Input Statements (Page 2 of 2)

JAN 15, 1995		SHORT ITEM REPORT				PAGE 1
PART NUMBER	PART DESCRIPTION	QUANTITY ON HAND	QUANTITY ON ORDER	MINIMUM BALANCE	SHORT	
10020	WIDGET DIGITS	30		60	30	
10040	WRENCH WIDGETS	15		25	10	
10050	HOLLOW WIDGETS			40	40	
10060	GIRL WIDGETS	25	25	80	30	
10070	BOY WIDGETS			80	80	
10090	YELLOW WIDGETS			90	90	
10100	GREEN WIDGETS			80	80	
10110	STEERING WIDGETS			70	70	
10130	WIDGET PILLS			200	200	
10140	LIGGIT WIDGETS	18	945	1,800	837	
10160	TALL WIDGETS			2,550	2,500	
10180	MEDIUM SIZED WIDGETS		2,500	4,500	2,000	
10190	LONG WIDGETS	4,790		5,500	710	
10200	WIDE WIDGETS			6,500	6,500	
GRAND COUNT	14					

Figure 7-4 Sample Run Output Report

Job Control Language Requirements

VISION:Builder Batch Free-Form Input, which runs as a standard job step under OS/390, requires one input data set and generates two output data sets. The following table shows the ddnames for the three files used by this capability. All three files have a fixed length record format and a logical record length of 80-bytes. [Figure 8-1](#) shows a typical OS/390 job stream using this capability.

ddname	Description
M4FREE	Free-form source input statements.
M4INPUT	Fixed form VISION:Builder output statements. This file can be used directly as a VISION:Builder source statement input file.
FREELIST	List of free-form source input statements and diagnostic messages.

Notes

```

1 //jobname JOB job card parameters
  //JOBLIB DD DSN=your.builder.loadlib,DISP=SHR
  //stepnam1 EXEC PGM=FORMATER
  //FREELIST DD SYSOUT=A
  //M4INPUT DD UNIT=SYSDA,DISP=(,PASS),
  //          SPACE=(TRK,(2,2),RLSE),
  //          DCB=BLKSIZE=6160
  //M4FREE DD *
          .
          (free-form source statements)
          .
          .
          .
  //stepnam2 EXEC PGM=MARKIV,COND=(8,LE,stepnam1)
  //M4INPUT DD DSN=*.stepnam1.M4INPUT,DISP=OLD
          .
          .
          .
          (remainder of standard VISION:Builder JCL)

```

Figure 8-1 A Sample OS/390 Job Stream

The numbered statement in [Figure 8-1](#) is explained as follows:

Note	Explanation
1	EXEC PGM=BFI can be used instead of EXEC PGM=FORMATER.

Using the VISION:Builder Executive Under TSO

The VISION:Builder Online Executive (OLX) option combines the capabilities provided by the IBM Time Sharing Option (TSO) for program execution with VISION:Builder source input and formatted output. OLX acts as the monitor for VISION:Builder online activity.

OLX includes the ability to access an online option of VISION:Builder: Online Free-Form Input (OFI).

OFI is a comprehensive free-form language preprocessor that adapts VISION:Builder to an online environment. OFI is discussed further in this chapter.

A VISION:Builder online session is initiated, after logging onto TSO, by entering the M4EXEC command. M4EXEC calls the monitor program for execution. It also makes the OLX commands and all their functions available to you. All TSO commands except TEST are accessible while M4EXEC is active.

Input and Output Devices

You can use any input or output device available to standard VISION:Builder batch operations; also, your terminal can be used as an I/O device. The terminal can be used as an input file, for instance, to enter transaction data to update a master file. Existing data sets, whether they were created online or in batch runs, can be accessed by VISION:Builder. If your output is a report, the terminal can be used as an output device. Through the use of TSO commands, the report can be routed to your system printer.

Furnishing Allocation Data

The standard JCL used with the batch version of VISION:Builder is not required with OLX. However, allocation specifications (such as DS names) must still be furnished for all the files to be used during a VISION:Builder session. File allocations can be supplied either through use of the TSO ALLOCATE command or by the data set operands of the EDITIV (if OFI is used), RUNIV, LIB, and SUBIV commands.

All VISION:Builder files allocated using the ALLOCATE command must be given a file name that corresponds to the ddname shown in the table on page 9-2. If the file has not been allocated, OLX prompts you for the data set name. Once you enter the data set name, the file is automatically allocated. If you enter an asterisk (*) in response to the prompt, the terminal becomes the file. If a null line is entered, the file is allocated as a dummy file.

Description	ddname	VISION:Builder ddname Assignments/Comments
Source Input	M4INPUT	Assigned to a standard I/O device which is: <ol style="list-style-type: none"> 1. physical sequential. 2. blocked one or more 80-character records per block.
Object Input	M4OWN	Must be a load module in a partitioned data set.
Source Listing and Reports	M4LIST	Usually assigned to a printer, but can be any data set that is: <ol style="list-style-type: none"> 1. physical sequential. 2. blocked one or more 133-character records per block (or the M4PARAMS value).
Alternate List File	M4LIST1 (or any unique name)	Any data set that is: <ol style="list-style-type: none"> 1. physical sequential. 2. blocked one or more 133-character records per block (or the M4PARAMS value).
Old Master In	M4OLD	A physical sequential, indexed sequential, VSAM data set, or IMS database.
New Master Out	M4NEW	A physical sequential, indexed sequential, VSAM data set, or IMS database. ⁴
Transaction In	M4TRAN	A physical sequential, indexed sequential, VSAM data set, or IMS database. ⁴
Audit File Out	M4AUDIT	A physical sequential dataset. ⁴
Report File Out	M4REPO	A physical sequential dataset.
Report File In	M4REPI	A physical sequential dataset.
Coordinated File n	M4CORDn	A physical sequential, indexed sequential, or VSAM data set (n=1-9).

Description	ddname	VISION:Builder ddname Assignments/Comments
Subfile n	M4SUBFn	A physical sequential dataset (n=0-9).
Sort Control	M4SORT	Any data set which is: <ol style="list-style-type: none"> 1. physical sequential. 2. blocked one or more 80-character records per block.
Common Library	M4LIB	A direct-access data set. DISP=SHR may be specified on the DD statement for M4LIB; if this is done, VISION:Builder will ensure the integrity of the library. Multiple common libraries are supported for processing runs only. In this case, the ddname is M4LIBn where n is a number 1-9.
Rejected Transaction File	M4REJCT	Must be a physical sequential data set. ⁴
Alternate Report File Out	M4REPN	A physical sequential data set (n = 2-9).
Work File	M4WORK	A physical sequential data set.
Source Statement Out	M4SSOUT	Assigned to a standard I/O device that is: <ol style="list-style-type: none"> 1. physical sequential. 2. blocked one or more 80-character records per block.
Program Analyzer	M4PAOUT	A physical sequential data set.
Checkpoint File	M4CHKn	A file containing checkpoint information (n = 1-2).
Report Summary File	Any unique name	A physical sequential data set, blocked with variable length records.
Report Manager Reports	M4PRINT (or any unique name)	Usually assigned to a printer, but can be any data set that is: <ol style="list-style-type: none"> 1. physical sequential. 2. blocked one or more 133-character records per block (or the M4PARAMS value).

Using CLISTs

The TSO EDIT command can be used to create a data set of TSO commands known as a command list (CLIST). The TSO EXEC command is used to call the CLIST into main storage for execution. Each command in the data set is executed in the sequence where it occurs in the CLIST. The CLIST can be composed of any valid combination of TSO and OLX commands plus data lines.

Messages Issued By OLX

OLX provides the following types of messages:

- prompting
- information
- error

OLX prompting messages are similar to TSO prompting messages. Information messages include all messages that are issued at your request plus all messages that are issued by the system and which expect no response. Error messages include the initial indication of the error.

OLX Commands

You communicate with OLX through the use of commands for OLX to perform certain functions. These commands are:

Command	Description
EDITIV	The EDITIV command calls in the OFI command processor.
END	The END command terminates a VISION:Builder online session.
HELP	The HELP command requests information about the other OLX commands.
LIB	The LIB command specifies the name of the common library and the function that is to be performed upon it.
M4EXEC	The M4EXEC command initiates a VISION:Builder online session and makes the other OLX commands available for use.
QUIT	The QUIT command terminates a VISION:Builder online session.
RUNIV	The RUNIV command invokes VISION:Builder for execution.
SUBIV	The SUBIV command controls VISION:Builder execution and provides access to IMS databases.

Data Set and Command Security Interfaces

You can invoke data set security routines directly from the TSO Dynamic Allocation Interface Routine (DAIR). Alternatively, you can perform security checking on any data set that is allocated by the RUNIV, SUBIV, EDITIV, M4EXEC, and LIB functions of OLX through the use of a data set security exit routine that you can alter.

The security exit routine allows you to check the data set names each time the DAIR routine is called and to supply a return code to OLX indicating whether or not the DAIR function is to be performed for the data set. It is, therefore, possible to prevent access to designated data sets.

You can also implement command security procedures by using the OLX command security exit. The command security exit routine, M4EXECCE, is called by OLX before each command or implicit CLIST is invoked. M4EXECCE can examine the command name and determine whether you are allowed to issue the command. If you are not allowed to issue the command, OLX issues a message to the terminal and terminates any active command or CLIST.

Online Query Language

Online Query Language (OQL) is a Query Language preprocessor that generates the VISION:Builder input statements to perform processing runs and is an available option under the CMS operating system.

Online Free-Form Input

Online Free-Form Input (OFI) is a comprehensive free-form language preprocessor for VISION:Builder that brings all of the VISION:Builder capabilities to the terminal.

OFI is called by OLX through the EDITIV command. You remain in OFI until you terminate the command processor by one of three commands: END, FILE, QUIT.

Input Mode and Edit Mode

To create and edit stacks of data, OFI operates in either input or edit mode. If the stack is identified as new, OFI automatically places itself in input mode after printing the message NEWFILE. If the stack is previously filed, OFI places itself in edit mode.

In the input mode, VISION:Builder source statements can be typed and entered by pressing the carriage return key. The input mode is in effect until a null line is entered. You can also switch to input mode by entering the command INPUT.

After entering a null line, OFI is in edit mode and informs you of the change in mode by printing the message EDITIV. In the edit mode, you can enter any of the OFI commands that point to particular lines of the stack, modify lines of data, add or delete lines of data, or control editing of input.

Input of VISION:Builder Source Statements

Input of VISION:Builder source statements to OFI is made field by field. Identification of fields can be achieved in several ways:

- By specifying the keyword associated with a field.
- By using an abbreviated version of the keyword.
- By specifying the actual column position where the field begins in the VISION:Builder fixed form.
- By using commas to delineate the fields' relative positions on the fixed format form.

OFI scans the input statement until the end of the statement is encountered and the next input line is read. If the character preceding the end of line is a comma, OFI considers the statement incomplete and the next input line is considered a continuation of the previous line.

[Figure 9-1](#) lists OFI keywords and their abbreviations for each VISION:Builder form type.

```

AA
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -AA-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
SEQUENCE     S                TEXT         T          DECKID       D
*REQUEST     R
AD
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -AD-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
DELETE       DEL         GLOSSARY     G          FORMAT       O
RECSIZE      RE         BLKFACTR     L          BUFSIZE      BU
ROWSIZE      RO         COLSIZE      C          DECKID       DEC
*FILENAME    FI
BA
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -BA-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
*FIELD       FIE        SRCHTYPE     S          TABLE.      T
ARGUMENT     A          DECKID       D          *FILENAME    FIL
BN
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -BN-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
*FIELD       FIE        HEADING      H          DECKID       D
*FILENAME    FIL
B0
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -B0-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
*FIELD       FIE        LOCATION     LO         LENGTH       LE
TYPE         T          KEY          K          ROUND        R
DPLACES     DP         EDFLOAT      EDFL       EDFIL        EDFI
EDTRAIL     EDT        EDLENGTH     EDL       DECKID       DE
*FILENAME    FIL
    
```

Note: Keywords marked with an asterisk (*) are required.

Figure 9-1 Keywords Available for VISION:Builder Form Types (Page 1 of 5)

```

CP
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -CP-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
CKPTFILE     CK          ALTERNAT     A          SORT          S
TIME         T           COUNT        COU         CONTROL       CON
OPERATOR     O           EOVS         E           DECKID        D
*RUNAME      R           CHECKID      CH
CR
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -CR-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
DATE         DA          ID           I           DECKID        DE
*REQUEST     R
CT
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -CT-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
OPERATIO     O           NAME1        NAME1       NAME2         NAME2
NAME3        NAME3      NAME4        NAME4       NAME5         NAME5
NAME6        NAME6      CONTINUE     C           DECKID        D
REQGROUP     R
EN
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -EN-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
SUMMARY      SUM         SPACING      SP          P8LPI         P8
WIDTH        W           HEIGHT       H           IMAGES        IM
FORMS        F           MAXLINES     MAXL        MAXPAGES      MAXP
BOTTOM       BOT         HEADTYPE     HEADT       HEADPOS       HEADP
DATEPOS      DA          PAGEPOS      PAGEP       PAGENO        PAGEN
LINENO       LI          LABELS       LA          REPORT        REPO
ENTIRE       EN         SUBFILE      SUB         BLOCK         BL
RECFM        REC         BORDER       BOR         EMPTYCTL     EM
AUTOGRND     AU         SINGLETL     SI          REPEATSB     REPE
INVSUMID     N          DECKID       DE          *REQUEST     REQ
CONTINU      C
ER
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -ER-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
DATE         DA          ID           I           MAXSEL        M
SELECT       SEL         SUMMARY      SUM         SPACING       SPA
FORMS        F           WIDTH        W           HEIGHT        H
LINENO       L           TYPE         T           SPECIAL       SPE
SETNAME      SET         SCREEN       SC          BRANCH        B
REINIT       REI        PARALOOP     P           DECKID        DE
*REQUEST     REQ
FD
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -FD-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
IDENT        D           DELETE       DEL         GLOSSARY      G
FORMAT       FO          RECSIZE     R           BLKFACTR      BL
BUFSIZE      BU         MAPPED      MAPP        INIT          IN
TERM         T           MAPRECSZ    MAPR        DECKID        DEC
*FILENAME    FI
FN
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -FN-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
SEQUENCE     S           TEXT         T           DECKID        D
*REQUEST     R
IT
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -IT-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
GENERIC      G           TYPECODE     T           ITEMNAME      I
MASTER       M           EXPDATE     E           PERIOD        P
UPDATER      U           DECKID       D           RUNAME        R
LA
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -LA-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
*FIELD       FIE        DELETE       DEL         SRCHTYPE      S
TABLE        T           ARGUMENT     A           DECKID        DEC
*FILENAME    FIL

```

Note: Keywords marked with an asterisk (*) are required.

Figure 9-1 Keywords Available for VISION:Builder Form Types (Page 2 of 5)

```

Jn
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -Jn-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
*GRAPHTYP    G                SUPPRESS     SU          BARTYPE      BART
BARWIDTH     BARW              PRMCHAR      P           SCDCHAR      SC
FITCHAR      F                HZACHAR      HZA        HZHCHAR      HZH
VTACHAR      VTA              VTHCHAR      VTH        DECKID       D
*REQUEST     R

Kn
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -Kn-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
*GRAPHMOD    G                START        S           END          E
DELTA        DEL          DECKID       DEC        *REQUEST     R

LM
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -LM-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
*NAME        N                DELETE       DEL        *COMMAND     F
REQUEST      R                DECKID       DEC        *FILENAME

LN
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -LN-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
*FIELD       FIE          DELETE       DEL        HEADING      H
DECKID       DEC        *FILENAME    IL

LS
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -LS-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
NAME         N                DELETE       DEL        SEGMENT      S
LEVEL        L                ORDER        O           DECKID       DEC
*FILENAME    F

L0
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -L0-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
*FIELD       FIE          DELETE       DEL        SEGMENT      S
LEVEL        LEV          LOCATION     LO         LENGTH       LEN
TYPE         T            KEY          K           ROUND        R
DPLACES     DP           COUNTSEG    C           OCCURS       O
EDFLOAT     EDFL        EDFILL       EDFI        EDTRAIL      EDT
EDLENGTH    EDL         DECKID       DEC        *FILENAME    FIL
ALIAS       A

PA
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -PA-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
CNVMSG       C            XREF         X           TRACE        T
MAXTRACE     M            BLKSIZE      B           FORMAT       F
DECKID       D            *RUNAME      R

OC
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -OC-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
MODULE        M            INTERFAC    IN          IDENTIFY     ID
EXIT1        EXIT1       EXIT2       EXIT2       EXIT3        EXIT3
EXIT4        EXIT4       EXIT5       EXIT5       EXIT6        EXIT6
EXIT7        EXIT7       EXIT8       EXIT8       EXIT9        EXIT9
EXIT10       EXIT10      DECKID      D           *RUNAME     R

PN
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -PN-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
SEQUENCE     S            TEXT        T           DECKID       D
*REQUEST     R

PR
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -PR-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
SEQUENCE     S            LEVEL       L           CONNECTO     C
AQUAL        AQ           AFIELD      AF          *OPERATOR    O
BQUAL        BQ           BFIELD      BF          RQUAL        RQ
RFIELD       RF           PSTART     PS          PNUMBER      PN
POPERAND     PO          DECKID      D           *REQUEST     RE

```

Note: Keywords marked with an asterisk (*) are required.

Figure 9-1 Keywords Available for VISION:Builder Form Types (Page 3 of 5)

```

RC
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -RC-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
FILENAME     F              OLDIN        O          NEWOUT      N
SEQUENCE     SE            RANDOM       RA         UPDATE      U
TRANFILE1    T            REPORT       REP        AUDIT       AU
BUFFERS      B            SORT         SO         DELIMIT     DEL
SCAN         SC           SRCLIST      SR         SPOOL       SP
REJECT1    REJ          COROPT       C          MSGOPT      M
RPTOPT       RP           SSROUT       SS         APPLTYPE   AP
START        ST           END          E          DECKID      DEC
*RUNAME      RU

RF
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -RF-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
*FILENAME    F          LABEL        L          DDNAME      DD
KEY1         KEY1       KEY2         KEY2       KEY3        KEY3
ICF          I          CRD          C          DBI         DB
MODULE       M          ARRAY        A          STORAGE     S
PASSWORD1  P          DECKID       DE         *RUNAME     R

RG
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -RG-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
GROUP1       GROUP1     GROUP2       GROUP2     GROUP3      GROUP3
GROUP4       GROUP4     GROUP5       GROUP5     DECKID      D
*RUNAME      R

RN
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -RN-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
SEQUENCE     SE         SPACES       SP          QUAL         Q
*FIELD       F          ENDLINE      EN          NONPRINT     N
SORT         SO         DESCEND      DES         CONTROL      CON
SUBTITLE     SU         TOTAL        T          CUMULATE     CU
COUNT       COU        MAXIMUM      MA          MINIMUM      MI
AVERAGE     A          GRAPHMOD     G          DPLACES      DP
PRQUAL       PRQ        PRFIELD      PRF         RATIO        RA
PERCENT      PE         EDIT         ED          PSTART       PS
PNUMBER      PN         DECKID       DEC         *REQUEST     RE

RP
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -RP-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
*PNAME       PN         PVALUE       PV          DECKID      D
*RUNAME      R

TB
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -TB-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
TYPE         TY         DELETE       DEL         PRINT        P
ALENGTH      AL         ATYPE        AT          ADPLACES    AD
RLENGTH      RL         RTYPE        RT          RDPLACES    D
DECKID1    DEC        *TABLE       TA

TD
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -TD-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
ID1          ID1        LOC1         LOC1        LEN1         LEN1
ID2          ID2        LOC2         LOC2        LEN2         LEN2
DELETE       DEL        GLOSSARY     GL          *MASTER     M
CREATE       C          INSERT       IN          FORMAT       F
RECSIZE     R          BLKFACTR     BL          BUFSIZE     BU
DECKID      DEC        *GROUP       GR

TE
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -TE-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
CONTINUE     C          DELETE       DEL         ARGUMENT     A
RESULT       R          DECKID       DEC         *TABLE       T

TF
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -TF-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
*FIELD       F          LENGTH       L          TYPE         T
DPLACES     DP         EDFLOAT      EDFL        EDFILL       EDFI
EDTRAIL     EDT        EDLENGTH     EDL         INITIAL      I
HEADING1    HEADING1   HEADING2     HEADING2    DECKID      DE
*REQUEST     R

```

Note: Keywords marked with an asterisk (*) are required.

Figure 9-1 Keywords Available for VISION:Builder Form Types (Page 4 of 5)

```

TL4
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -TL-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
ID1           ID1           ID2           D2          DELETE       DEL
*LOCATION      LO            *LENGTH      LE          TYPE         T
*NAME        N            *ACTION      A           DPLACES     DP
MINIMUM      MI           MAXIMUM      MA          PATTERN     P
VALIDTYP     V           DECKID       DEC         *GROUP      G
TN
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -TN-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
SEQUENCE     S           TEXT         T           DECKID       D
*REQUEST     R
UC
THE FOLLOWING KEYWORDS ARE AVAILABLE FOR FORM TYPE -UC-:
KEYWORD      ABBR.      KEYWORD      ABBR.      KEYWORD      ABBR.
OPCODE       O           GENERIC      G           TYPECODE     T
ITEMNAME     M           MASTER       M           DDNAME       DD
REPLACE      RE          RULE         RUL        BLOCKING     B
PURGE        P           DECKID       DE          RUNAME       RUN
    
```

Note: Keywords marked with an asterisk (*) are required.

Figure 9-1 Keywords Available for VISION:Builder Form Types (Page 5 of 5)

OFI Commands

The following table lists OFI commands, the minimum abbreviation that can be used, and a summary of the command function, in alphabetical order.

Command	Abbreviation	Function
AGAIN	A	Reuses previous LOCATE or FIND command.
ALTER	A	Alters contents of a field.
BOTTOM	B	Moves pointer to last line.
BRIEF	BR	Limits system printed responses.
CHANGE	C	Modifies contents of a line.
DELETE	DE	Deletes lines from stack.
DOWN	D	Moves pointer down.
END	E	Terminates OFI.
FILE	FILE	Writes stack and terminates OFI.
FIND	F	Finds a line in stack.
GET	GE	Inserts saved lines in data set.
GLOSSARY	GL	Lists valid field names for specified file.
HELP	H	Lists information about OFI commands.
INPUT	I	Places OFI in input mode.
INSERT	I	Inserts line of data.

Command	Abbreviation	Function
KEYWORD	K	Lists keywords for a given VISION:Builder form.
LINENO	LI	Prints current line number.
LIST	LIST	Prints current line.
LOCATE	L	Locates a line.
NEXT	N	Moves pointer down.
PRINT	P	Prints lines from stack.
PUT	PUT	Stores lines in a temporary buffer.
PUTD	PUTD	Stores deleted lines in a temporary buffer.
QUIT	Q	Terminates OFI.
REPLACE	R	Replaces line.
SAVE	SAVE	Writes stack.
SCAN	SC	Scans stack for errors.
TOP	T	Moves pointer to top of stack.
UP	U	Moves pointer up.
USE	US	Specifies VISION:Builder file definition to validate field names.
VERIFY	V	Revokes BRIEF command.
?	?	Prints additional information.

Sample Run

[Figure 9-2](#) illustrates an example of VISION:Builder free-form language. The example starts from the point when OLX is invoked and goes through each step until the request is filled.

```
m4exec
BEGIN MARKIV SESSION - 10/15/92 17.57.36
MARKIV: editiv
NEWFILE.
INPUT:
rc run=demorun,file=caseprob,oldin=s,report=s
er request=demoreq,date=today,width=c
pr af=qanonord,,qanonhnd,t,total
pr afield=min-bal,operator=gt,bqual=t,bfield=total
pr af=min-bal,,t,short,t,short
r1 f=partno,sort=1,count=g
r1 f=descript
r1 f=qanonhnd
r1 f=qanonord
```

Figure 9-2 VISION:Builder Free-Form Input Sample (Page 1 of 2)

```

r1 f=minbal
r1 q=t,short
t1 text='short item report#'

EDITIV
file
SORTED
ENTER SAVE DATASET NAME:
job
FILED
MARKIV:
    
```

Figure 9-2 VISION:Builder Free-Form Input Sample (Page 2 of 2)

The output of this request is shown in [Figure 9-3](#).

JAN 15, 1995		SHORT ITEM REPORT			PAGE 1	
PART NUMBER	PART DESCRIPTION	QUANTITY ON HAND	QUANTITY ON ORDER	MINIMUM BALANCE	SHORT	
10020	WIDGET DIGITS	30		60	30	
10040	WRENCH WIDGETS	15		25	10	
10050	HOLLOW WIDGETS			40	40	
10060	GIRL WIDGETS	25	25	80	30	
10070	BOY WIDGETS			80	80	
10090	YELLOW WIDGETS			90	90	
10100	GREEN WIDGETS			80	80	
10110	STEERING WIDGETS			70	70	
10130	WIDGET PILLS			200	200	
10140	LIGGIT WIDGETS		945	1,800	837	
10160	TALL WIDGETS			2,500	2,500	
10180	MEDIUM SIZED WIDGETS		2,500	4,500	2,000	
10190	LONG WIDGETS	4,790		5,500	710	
10200	WIDE WIDGETS			6,500	6,500	
GRAND	COUNT	14				

Figure 9-3 Sample Report Run

JCL Override Parameters

You can override the record format, buffering techniques, buffer or block size, and record size indicated on the VISION:Builder file or transaction definitions. The override specifications are made in the DCB subparameters of the DD statement according to the following rules. However, override specifications for the common library (M4LIB) are not valid and are ignored if used. The block size must match M4PARAMS or an ABEND may occur. In the three-step processing run, the M4PARAMS value is used regardless. The number of buffers can be overridden for all files.

Note: The DCB should always be omitted from M4REPO, SORTIN, SORTOUT, and M4REPI.

Rule 1

The logical record size (LRECL) can be overridden for F or V formats. If the format on the file definition is F and the record size does not agree with the size in the FD or TD statement, a message is issued. If the format is V on the file definition and the LRECL from the header label/DD statement is smaller than the record size+4 on the glossary, a warning message is issued. The override size is accepted.

The logical record size (LRECL) cannot be overridden for report files (M4REPO, M4REPI, M4REPn).

Rule 2

The block size (BLKSIZE) can be overridden for all file types.

If block size is overridden for fixed length files (VISION:Builder F and I formats), it is checked for being an integral multiple of LRECL. If it is not, a message is printed and the job is terminated.

If block size (buffer size) is overridden for variable length files (V format in VISION:Builder), it is checked for being at least 4-bytes greater than LRECL. If the block size is less than LRECL+4, the block size is adjusted to LRECL+4.

The block size specified for report files (M4REPO, M4REPI, M4REPn) must never be smaller than the block size specified in M4PARAMS.

If you are running under an OS/390 system that supports the system-determined block size capability, you can override the block size for output files that qualify for this capability. An override to a non-zero value turns the capability off for this file and sets the block size to the specified value. An override to a zero value or not specifying a block size at all, turns the capability on for this file and the system will determine an appropriate block size.

Rule 3

Any specification in the record format (RECFM) field can be overridden, except the actual record format (first entry). Therefore, the main function of the RECFM override is to specify the B code (blocking) and S code (standard or spanned) blocks. If the RECFM code is supplied, the complete format must be supplied (for example, FBS).

Example 1

In an FD statement, the following is specified:

- RECORDS PER BLOCK 2
- RECORD SIZE 100
- RECORD FORMAT F

This results in VISION:Builder assigning:

RECFM = FBS (VISION:Builder assumes standard blocks.)

Example 2

In an FD statement, the following is specified:

- RECORD FORMAT V
- BUFFER SIZE 5000

This results in VISION:Builder assigning:

RECFM=VB

If you want to process a file with spanned records, specify:

RECFM=VBS

This extension does not relieve you of the responsibility for entering a reasonable value of buffer size on the FD or TD forms.

Rule 4

The number of buffers (BUFNO) can be overridden for all files.

Rule 5

Optional services (OPTCD) can be specified within OS/390 system restrictions.

Concatenation of Input Data Sets

You can concatenate like or unlike sequential input or update-in-place data sets for processing by VISION:Builder (refer to the *IBM MVS Data Management Services Guide* for information regarding the definition of unlike concatenated data sets). When processing a concatenated data set with VISION:Builder, the following rules apply to the data sets specified on the second through last DD statements of the concatenation. If any one of these rules is violated, the run is terminated.

Note: Alternate report files each require their own sort and report steps.

Rule 1

The actual record format (types F, V, U) must be the same as that of the first data set in the concatenation.

Rule 2

If you are running under an OS/390 system that supports the system-determined block size capability, you can concatenate input data sets in any order. Otherwise, the block size must be less than or equal to that specified for the first data set.

If the first data set has a smaller block size than a later one, a DD override of the form `DCB=BLKSIZE=n` must be used on the first DD statement, where `n` is the largest block size in the concatenation.

If the record format of the first data set is FBS and the block size must be overridden, specify `DCB=(RECFM=FB,BLKSIZE=n)` to avoid premature end-of-file indications.

Rule 3

For record format F, the logical record length (LRECL) must be the same as for the first data set and the block size must be a multiple of the logical record length.

Rule 4

For record format V, the logical record length must be less than or equal to the LRECL for the first data set. If the block size is less than `LRECL+4`, it is adjusted to `LRECL+4`.

If any data set in the concatenation has spanned records, you must specify a maximum records size of at least 32K in the FD buffer size entry for the file. This activates true VBS processing.

Linking to VISION:Builder

This section describes the interfaces between VISION:Builder and the program that invokes VISION:Builder. It is included for OS/390 users who want to write a program that calls or links to VISION:Builder rather than executes VISION:Builder directly using the `// EXEC JCL` statement.

Calling Sequence

- Register 13 Save area where VISION:Builder saves the calling program's registers using the SAVE macro.
- Register 14 Return address used by VISION:Builder to exit to the calling program.
- Register 15 Entry point address in VISION:Builder.
- Register 1 Address of an input parameter list that specifies PARM information and a ddname list.

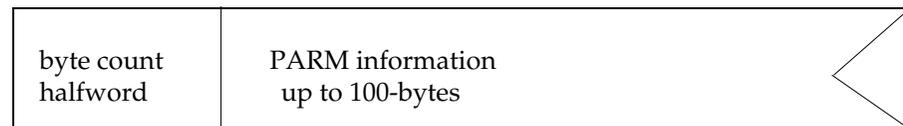
VISION:Builder Input Parameter List

The input parameter list supplied by you must contain one or two fullword addresses that are aligned on a fullword. The high order byte of each word, except for the last word, must be zero. The high order byte of the last word must contain X'80'; the ddname list can be omitted if the high order byte of the address of the PARM information is X'80'.

Location	← 1 Bytes →	← 3 Bytes →
0	X'00'	Address of PARM information
4	X'80'	Address of ddname list

PARM Information

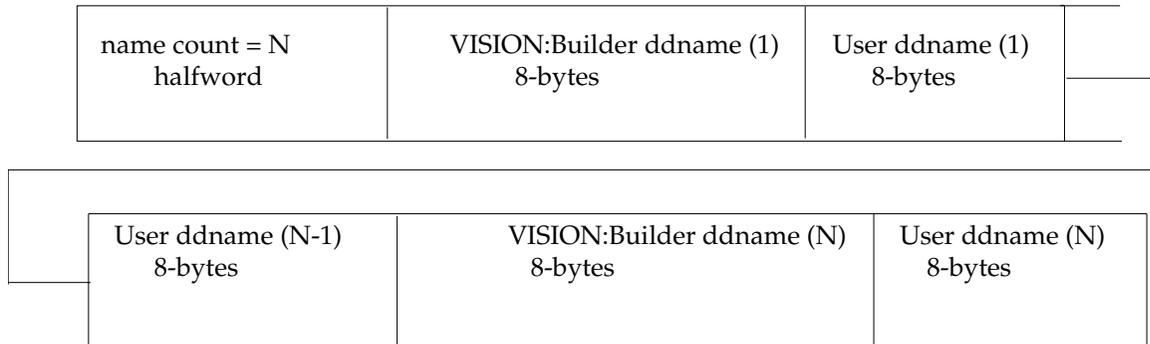
PARM information has the format used by OS/390 to pass PARM data on the EXEC statement to VISION:Builder. You can use the address to pass PARM information to own code routines using hook 10. The address of the PARM information points to a halfword count field:



If there is no PARM information, the parameter list address should be zero or the address of a halfword of zero.

ddname

The ddname list contains information used by VISION:Builder to convert VISION:Builder ddnames to your ddnames. Any names in the list that are not VISION:Builder ddnames are ignored by VISION:Builder. The list has the following format:



When VISION:Builder encounters one of its ddnames in the list, the ddname immediately following is used instead of the standard VISION:Builder name. Note that the ddnames M4LIB, M4CHK1, and M4CHK2 cannot be changed.

If a ddname list is not provided, the parameter list address should be zero or the address of a halfword of zero.

Exit Sequence

VISION:Builder closes files, deletes loaded modules, frees main storage, and restores registers before returning to your return address supplied in Register 14 on entry. A condition code is returned to you in Register 15.

Sort Control Statements

VISION:Builder produces sort control statements for report file sorting when requested to do so (by means of the Sort Control specification on the RC statement). These control statements are output to M4SORT and listed on M4LIST. Only one set of control statements is produced per run regardless of the number of report files created; however, the same sort control statements should be used for all report files in the run.

The actual sort control statements generated depend on the sort program specified in M4PARAMS. The sort control statements are shown below in [Figure 10-1](#) and apply to Sort Program 5740-SM1.

Notes

```

1   SORT FIELDS=(5,***,A),FORMAT=BI,FILSZ=[E]*****[,CKPT],NOEQUALS
2   RECORD TYPE=V,LENGTH=(*****,*****,*****,*****,*****)
3   [MODS E35=(MARKSS,****, M4SRTLIB,N)]
    END

```

Figure 10-1 Sort Control Statements

Notes		Explanation	
1	SORT	FIELDS =	The length of the sort control fields.
		FILSZ =	E is an optional entry that is used if multiple report files are specified. The number of records on the report file (or the maximum of all report file counts) is inserted.
		CKPT =	An optional entry that is used if sort checkpoints are requested on the CP statement.
2	RECORD	LENGTH =	Inserts 1-5: <ol style="list-style-type: none"> 1. The report file record size (LRECL). 2. The maximum record length. 3. The report file record size (LRECL). 4. Minimum record length. 5. The modal record length (size of maximum possible data record is the value used).
3	MODS	E35 =	Optional information used only if Report File Optimization is requested; the size of the MARKSS module in bytes is inserted.

Limiting the Number of Records Read During Input File Processing

There are instances when the number of records read from an input file could be less than the total number of records on the file.

- The start search specification can be used in conjunction with any master file with direct-access capabilities, when the file is being processed sequentially. VISION:Builder retrieves the first master file record with a key value equal to or greater than the start search specification and begins standard sequential processing.
- The end search specification limits records read from the old master file. The run is terminated after processing the last record of a key less than or equal to the specified key, or the next higher one if the specified key is non-existent.
- The direct-read specification limits records read from an ISAM or VSAM old master file by the key value specified.
- The ICF capability limits records read from an ISAM or VSAM coordinated file by the key value specified on the RF statement.
- The EOF flag can be set to E to limit the number of records read from any input file.
- Standard coordination limits records read to those records in the coordinated file that match to a record on the master file.
- With chained coordination, one coordinated file matches to the master file, another coordinated file matches to the first coordinated file, and so on. (This is different from unmatched coordination, in which all coordinated file records are read.)
- User read coordinated files can limit records read from any sequentially processed coordinated file by the use of an RD operator within your request.
- Runs that terminate with a type 4 or higher error could end input file processing before all records have been read.

In all cases, VISION:Builder run statistics for input files reflect the number of records read, not the number of records on the file.

Resource Optimization

Resource optimization gives you the choice of optimizing processing speed or main storage, whichever resource is most important to your application. Selection of the type of optimization is made in the Resource Optimization entry on the RC statement.

The following entries on the RC statement pertain to resource optimization.

- STORAGE? – Enter a Y to optimize main storage allocation. Main storage is conserved during processing. Leave blank or enter N to optimize processing speed. Extra main storage is used to improve processing speed.
- MESSAGE PROCESSING?– Enter a Y to make processing step error messages resident, thereby eliminating the I/O time normally required to load the messages from disk. This time savings is most significant for those runs that result in a high number of transaction error messages. Approximately 3000-bytes of additional main storage are utilized for these messages.
- REPORT FILE – Enter R for single-step processing of sort or no-sort runs. Enter S to request report file optimization.

The S specification is valid for summary only reports that request no summaries other than TOTAL, CUM, PERCENT, or RATIO. This entry allows a VISION:Builder routine to be invoked as a sort exit during the output of the sort. (An M4SRTLIB DD statement that points to the loadlib containing the VISION:Builder routine, MARKSS, must be included in the sort step JCL.)

The S specification then computes the required summaries prior to writing the output from the sort and normally only outputs a record when a control break occurs. This eliminates the I/O required to write detail records as output from the sort and to read detail records into the report step.

Approximately 4000-bytes of main storage are utilized during the sort step for this routine. This option is not available if any of the following conditions exist:

- No-sort run.
- A report is not a summary only report.
- The length of the data records on the report file is greater than 400-bytes.
- More than 40 summary and control fields are specified in one report.
- Summaries are specified on a character field.
- COUNT, MAX, MIN, or AVG are specified.

VISION:Builder considers each report separately in determining whether it qualifies for report file optimization. The first of several reports in a step can fail to qualify for optimization while a subsequent report is optimized.

Report file optimization terminates for a report if the frequency of control breaks or the number of interrupts due to arithmetic overflow substantially reduce the number of records being eliminated.

Access Methods

VISION:Builder uses the following access methods:

- VSAM Used for sequential or direct-access of files. VSAM is also used for the common library only if your system supports this.
- BDAM Used for the common library only if your system supports this.
- BISAM Used with an indexed sequential file when direct-read or update-in-place is specified.
- QISAM Used with an indexed sequential file when direct-read is not specified.
- QSAM Used with sequential file processing.

Blocking Factor for ISAM Files

VISION:Builder supports blocked ISAM files only. A blocking factor of 1 is allowed. All ISAM files created by VISION:Builder are blocked.

Non-VSAM Variable-Spanned Records

VISION:Builder supports variable-spanned record I/O. To activate VBS I/O processing, the variable-spanned file must be defined to VISION:Builder as a standard variable length file with a maximum record size of 32K or greater specified in the FD buffer size entry. The DD statement or header label for the file must specify RECFM=VBS. Existing variable-spanned files that have LRECL=BLKSIZE can be processed because VISION:Builder internally sets BLKSIZE=LRECL+4. If the VBS file record size is greater than 32760, specify LRECL=X in the DCB.

The report files: M4REPO, M4REPI, and M4REP2 through M4REP9 cannot be defined as variable-spanned.

Using VISION:Builder with DB2

This chapter applies only if you have the relational support option installed in your system.

This chapter presents the special requirements and considerations for using VISION:Builder to access or create DB2 databases. VISION:Builder can communicate with DB2 using any of three attach facilities provided with DB2:

- [TSO Attach Facility](#)
- [IMS Attach Facility](#)
- [CALL Attach Facility](#)

The choice of which facility to use depends upon the requirements of your particular application and your installation's standards.

TSO Attach Facility

The TSO Attach Facility can be used to access DB2 databases using VISION:Builder operating in either batch or online mode. Any file type or database except IMS databases can be accessed concurrently within your application when using VISION:Builder with the TSO Attach Facility. The sample JCL in [Figure 11-1](#) illustrates the use of VISION:Builder with the TSO Attach Facility.

```
//TSOAT  JOB
//step   EXEC  PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB DD  DSN=your.builder.loadlib,DIS P=SHR
//       DD  DSN=your.comlib.loadlib,DISP=SHR
//SYSTSPRT DD  SYSOUT =*
//SYSTSIN DD  *
           DSN SYSTEM(ssid)      where ssid=your DB2 subsystem id
           RUN PROGRAM(MARKIV)  PLAN(planname) LIB('your.builder.loadlib')
           END
//M4LIST  DD  SYSOUT=*
//M4LIB   DD  DSN=your.m4lib,DISP=SHR
//M4REPO  DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//M4INPUT DD  DSN=your.builder.program,DISP=SHR
```

Figure 11-1 Sample JCL Using the TSO Attach Facility

The plan name in the RUN statement above must correspond to the plan name specified when the DB2 access module MARKSQLT was pre-processed, compiled, and link edited with the TSO Attach Facility interface module DSNELI. This process should have been performed when VISION:Builder was installed at your installation. See the IBM book *IBM DATABASE 2 Application Programming and SQL Guide* for further information regarding the DSN and RUN command parameters. Additional DD statements must be provided for each non-DB2 VISION:Builder file as discussed in [Chapter 2, VISION:Builder Runs, Run Control, and Execution JCL](#).

IMS Attach Facility

The IMS Attach Facility must be used when your VISION:Builder application accesses IMS databases as well as DB2 databases. The JCL requirements are shown in [Figure 11-2](#). See the IBM manual *IBM DATABASE 2 Application Programming and SQL Guide* for further information regarding DB2 applications in an IMS/ESA environment.

```
//imsbatch JOB (accounting information)
//*
//IMSDLI PROC MBR=TEMPNAME,PSB=,BUF=7,
//          SPIE=0,TEST=0,EXCPVR=0,RST=0,PRLD=,
//          SRCH=0,CKPTID=,MON=N,LOGA=0,FMTO=T,
//          IMSID=,SWAP=,DBRC=,IRLM=,IRLMNM=,BKO=N
//GO      EXEC PGM=DFSRRCC00,
//          PARM=(DLI,&MBR,&PSB,&BUF,
//          &SPIE&TEST&EXCPVR&RST,&PRLD,
//          &SRCH,&CKPTID,&MON,&LOGA,&FMTO,
//          &IMSID,&SWAP,&DBRC,&IRLM,&IRLMNM,&BKO)
//          PEND
//*
//M4P     EXEC IMSDLI,MBR=DSNMTV01,PSB=yourpsb,REGION=1500K
//STEPLIB DD DSN=IMSVS.RESLIB,DISP=SHR
//          DD DSN=your.builder.loadlib,DISP=SHR
//          DD DSN=your.comlib.loadlib,DISP=SHR
//DFSRESLB DD DSN=IMSVS.RESLIB,DISP=SHR
//IMS      DD DSN=your.psplib,DISP=SHR
//          DD DSN=your.dbdlib,DISP=SHR
//IEFRDER DD DSN=your.chk1.imslog,DISP=(NEW,CATLG),
//          UNIT=SYSDA,SPACE=(TRK,(5,2),RLSE),
//          DCB=(RECFM=VB,BLKSIZE=1920,LRECL=1916,BUFNO=2)
//DDITV02 DD *
ssid,SYS1,DSNMIN10,,R,-,DLIBATCH,yourplan,MARKDLI
//DDOTV02 DD UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(TRK,(1,1)),
//          DCB=(RECFM=VB,LRECL=4092,BLKSIZE=4096)
//M4LIB    DD DSN=your.m4lib,DISP=SHR
//M4LIST   DD sysout=*
//M4REPO   DD DSN=your.m4repo,DISP=(NEW,PASS),
//          UNIT=SYSDA,SPACE=(TRK,(10,5),RLSE)
//M4SORT   DD UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(TRK,1)
//M4INPUT  DD DSN=your.builder.program,DISP=SHR
//database DD DSN=your.ims.database,DISP=SHR
//*
//SORT     EXEC PGM=SORT,REGION=1500K,COND=(0,NE,M4P.GO)
//SYSOUT   DD SYSOUT=*
//SORTIN   DD DSN=*.M4P.GO.M4REPO,DISP=(OLD,DELETE)
//SORTOUT  DD UNIT=SYSDA,DISP=(NEW,PASS),
//          SPACE=(TRK,(10,5),RLSE)
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
```

Figure 11-2 Sample JCL for IMS Batch with DB2 (Page 1 of 2)

```

//SYSIN      DD DSN=* .M4P.GO.M4SORT,DISP=(OLD,DELETE)
//*
//M4R        EXEC PGM=MARKIV,REGION=1536K,COND=(0,NE,M4P.GO)
//STEPLIB   DD DSN=your.builder.loadlib,DISP=SHR
//          DD DSN=your.comlib.loadlib,DISP=SHR
//M4LIST     DD SYSOUT=*
//M4REPI     DD DSN=* .SORT.SORTOUT,DISP=(OLD,DELETE)
//M4INPUT    DD DSN=your.builder.run.control.stmt,DISP=SHR
//*
//DB2LST    EXEC PGM=DFSERA10,COND=EVEN
//STEPLIB   DD DSN=IMSVS.RESLIB,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSUT1     DD DSN=* .M4P.GO.DDOTV02,DISP=(OLD,DELETE)
//SYSIN      DD *
CONTROL     CNTL K=000,H=8000
OPTION      PRINT
//

```

Figure 11-2 Sample JCL for IMS Batch with DB2 (Page 2 of 2)

CALL Attach Facility

The Call Attach Facility is an alternate DB2 access facility for DB2 applications not operating in the CICS/VS or IMS/ESA environments. The Call Attach Facility is used by VISION:Builder whenever an RP statement with the DB2 parameter name is included with your VISION:Builder source statements. The application plan name specified on the DB2 parameter statement must correspond to a plan name specified when the DB2 access module MARKSQL was pre-processed, compiled, and link edited with the Call Attach Language Interface module DSNALI. This process should have been performed when VISION:Builder was installed at your installation. No special JCL statements are required by the Call Attach Facility. The sample JCL in [Figure 11-3](#) illustrates the use of the Call Attach Facility with VISION:Builder.

```

//CALLATT JOB
//step EXEC PGM=MARKIV
//STEPLIB DD DSN=your.builder.loadlib,DIS P=SHR
//          DD DSN=your.comlib.loadlib,DISP=SHR
//M4LIST   DD SYSOUT= *
//M4LIB    DD DSN=your.m4lib,DISP=SHR
//M4REPO   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//M4INPUT  DD DSN=your.builder.program,DISP=SHR (with DB2 RP statement)

```

Figure 11-3 Sample JCL Using the Call Attach Facility

Additional DD statements must be provided for each non-DB2 VISION:Builder file as discussed in [Chapter 2, VISION:Builder Runs, Run Control, and Execution JCL](#).

Common Library Access and Utilities

This chapter describes the following information:

- [Using Multiple Common Libraries on page 12-2](#)
- [Using Common Libraries on Shared DASD on page 12-2](#)
- The following Common library utilities:
 - [VISION:Builder Release 14.0 / COMLIB Release 4.5 Characteristics and Notes on page 12-4](#)
 - [The Common Library Service Program on page 12-6](#)
 - [MARKINIT Common Library Initialization on page 12-16](#)
 - [MARKDUMP and MARKREST Common Library Dump and Restore on page 12-17](#)
 - [MARKCON Common Library Condense on page 12-20](#)
- [Using Cataloged Procedures and Requests on page 12-21](#)
- [Listing and Retrieving Common Library Items on page 12-22](#)

The common libraries (M4LIBs) are referred to by the I/O access mechanism used to process the data sets, BDAM or VSAM. The two types of common libraries (BDAM or VSAM) can be used together in the same run, if appropriate. The COMLIB subsystem can handle both types of common libraries concurrently.

In each section that describes a particular utility, examples for execution of the utility are shown.

The common library service program MARKUTIL is described in [The Common Library Service Program on page 12-6](#). This program should be used for all common library maintenance. The MARKINIT, MARKDUMP, MARKREST and MARKCON utilities are also supplied and can be used for common library maintenance; however, they may not be supplied in future releases. These utilities are only provided as an interim step until existing users can convert to MARKUTIL.

MARKINIT, MARKDUMP, MARKREST and MARKCON are described in [MARKINIT Common Library Initialization on page 12-16](#) through [MARKCON Common Library Condense on page 12-20](#).

You can also find information about cataloged procedures and requests and their maintenance in [Using Cataloged Procedures and Requests on page 12-21](#).

Listing glossaries and source statement retrieval are described in [Listing and Retrieving Common Library Items on page 12-22](#).

Using Multiple Common Libraries

You can access up to ten common libraries. JCL must be provided for each common library you want to access. A run parameter (RP) statement, with the MULTILIB parameter, is used to specify which common libraries are to be accessed, as well as the specific order in which they are accessed.

For example, a JCL procedure can reference many common libraries, but your application source statements can control which common libraries are to be used. This is valuable in terms of reducing common library search time. For VISION:Builder, you can also use the RP statement MULTILIB entry with the parameter value of OFF specified; this inhibits the multiple common library process. Your product accesses the primary common library (M4LIB) and any JCL that references multiple common libraries is ignored.

With VISION:Builder, any set of conditions indicating a common library update situation causes your product to access only the primary common library. Your product accesses the first M4LIBn, specified in the MULTILIB statements, as the primary common library and ignores the balance of the M4LIBs. Also, multiple common libraries cannot be used where:

- Transaction files are being used without an RG statement.
- CT statements are included.
- An RP MULTILIB OFF statement exists.

When searches are made through multiple common libraries and duplicate names exist for any definitions, procedures, or requests, the first matching name found is the one used. No attempt is made to search further or to inform you that duplicates exist. If multiple common libraries were searched, the product outputs an informational message stating they were searched. If this statement is not evident, multiple common libraries were not searched.

Using Common Libraries on Shared DASD

There are different levels of protection for common libraries when the common libraries are being shared by multiple systems. The following subsections describe the levels of protection for VSAM and BDAM common libraries.

Sharing Common Libraries on DASD

VISION:Builder issues a global ENQ macro instruction for the duration when the common library data set is open, whether its reserve bit is on or off in MARKLIBP. This protects the common library from concurrent updates from a single system (CPU).

If you have a common library on shared DASD, the RESERVE option in MARKLIBP can be used to protect the common library against concurrent updates from multiple systems (CPUs). When the MARKLIBP reserve bit is on, the system issues a hardware “lock” of the DASD where the common library resides. This reserves a device for use by a particular system and protects against concurrent updates from multiple systems.

VISION:Builder uses a QNAME of M4LIB and an RNAME the same as the cluster or data set name when performing ENQs.

The following table shows the reserve and share option protection for VSAM and BDAM common libraries.

		BDAM	VSAM
RESERVE	ON	GLOBAL ENQ & RESERVE	RESERVE & GLOBAL ENQ
	OFF	GLOBAL ENQ	GLOBAL ENQ

Note: For BDAM, GLOBAL ENQ is only local in scope (one CPU) unless your installation has GRS.

Global Resource Serialization (GRS), a component of OS/390, enables your installation to share symbolically-named resources either between units of work in a single system or between multiple systems in a loosely coupled environment. When you have GRS, you do not need RESERVE because the local ENQ issued by the product provides sufficient protection.

The advantage of GRS over the RESERVE macro alone is that instead of locking all of the DASD, GRS issues a global ENQ macro instruction to serialize only the particular data set to be protected.

Note: In all cases, QNAME = M4LIB and RNAME = DSN.

Based upon the above mechanisms, it is safe to specify DISP=SHR for any M4LIB and M4LIBn DD statement regardless of whether updating will occur. The common library will always be protected against simultaneous updates using the global ENQ or RESERVE even when DISP=SHR is used.

VISION:Builder Release 14.0 / COMLIB Release 4.5 Characteristics and Notes

This section describes changes that occurred in the VISION:Builder Release 13.5 release which apply to that release and continue to apply if you skipped that release or Release 13.8. If you are upgrading to VISION:Builder Release 14.0 from VISION:Builder Release 13.5 or VISION:Builder Release 13.8, this section may be ignored.

Internal M4LIB Format

The internal format used to store File Definitions and Array Definitions was enhanced and expanded to accommodate the Alternate Name and Field Description entries and allow for future enhancements. The basic internal architecture of the COMLIB Library data set (M4LIB) has not changed, only the internal format of new or updated file and array definitions cataloged using VISION:Builder Release 13.5 or later and COMLIB Release 4.5 have changed.

As a result of the internal format changes made to file and array definitions created or updated using COMLIB 4.5, the definitions will not be compatible with previous releases of VISION:Builder and COMLIB. The conversion of the definitions occurs automatically whenever an existing file or array definition is updated using VISION:Builder Release 13.5 or later and COMLIB Release 4.5.

Because of this incompatibility with previous releases, considerations must be addressed with regard to using releases of VISION:Builder and COMLIB prior to 13.5 and 4.5.

It is strongly recommended that you make a complete backup of all Common Library (M4LIB) data sets using your previous VISION:Builder and COMLIB releases (13.5 and 4.0 or prior). These backups should be retained until you have fully implemented VISION:Builder Release 14.0 and are satisfied that you will no longer be using the previous releases of VISION:Builder and COMLIB.

In order to convert file and array definitions that were cataloged using VISION:Builder Release 14.0 and COMLIB Release 4.5 back to a prior release, run a source statement retrieval using VISION:Builder Release 14.0 to obtain the definition source statements; then, recatalog these definition source statements using the prior release.

Common Library Utility Work File (M4WORK) Format

With VISION:Builder Release 13.5 or later and COMLIB Release 4.5, the format of the library utility work file data set (M4WORK) has been changed to significantly improve the performance of the Common Library DUMP, RESTORE and CONDENSE operations. The format is now VBS with a block size of 32760. This allows the library utilities to write complete items spanning blocks as needed, dramatically reducing I/O counts and processing time.

As a result of the work file format changes, the M4WORK DUMP files created using VISION:Builder Release 13.5 or later and COMLIB Release 4.5 are incompatible with releases prior to VISION:Builder Release 13.5. This means that DUMP files created by releases prior to VISION:Builder Release 13.5 cannot be read by VISION:Builder Release 14.0 and vice versa.

We strongly recommend that you make a complete backup of all Common Library (M4LIB) data sets using your previous VISION:Builder and COMLIB releases (13.5 and 4.0 or prior). Retain these backups until you have fully implemented VISION:Builder Release 14.0 and are satisfied that you will no longer be using the previous releases of VISION:Builder and COMLIB.

We also recommend that you make a complete backup of all Common Library (M4LIB) data sets using your new VISION:Builder Release 13.5 or later and COMLIB Release 4.5 system as soon as the new releases are installed.

Common Library Data Set (M4LIB) Compression

In COMLIB 4.5, which is delivered with VISION:Builder Release 13.5 or later, you can compress items stored in the Common Library (M4LIB). Because of the changes to the internal formats of File and Array Definitions as discussed in a previous section, these definitions require additional space on the Common Library (M4LIB). To alleviate the impact of these changes on your disk space utilization, COMLIB 4.5 may be enabled to compress objects whose size exceeds a specified threshold.

Aside from the disk space utilization consideration, you may want to enable COMLIB compression in order to reduce the I/O activity performed by COMLIB. The compression option and item size threshold value are controlled by modifying the appropriate parameters on the COMLIB MARKLIBP module.

Compression of items by COMLIB occurs when an item is cataloged. A COMLIB enabled for compression continues to correctly access uncompressed items such as definitions, tables, and requests. Only as items are recataloged with a COMLIB enabled for compression will any item be compressed. Conversely, the COMLIB component will successfully access compressed objects even though compression has been disabled.

An entire Common Library (M4LIB) may be compressed by performing a DUMP/RESTORE or CONDENSE using a COMLIB enabled for compression. Conversely, a Common Library may be uncompressed by performing a DUMP/RESTORE or CONDENSE using a COMLIB which has compression disabled.

The Common Library Service Program

This section illustrates the JCL statements used with the common library service program. MARKUTIL is provided with VISION:Builder. In addition to the common library service program, MARKINIT, MARKDUMP, MARKREST, and MARKCON are supplied with VISION:Builder. This section describes how to execute these programs.

Use MARKUTIL for all your common library maintenance. MARKUTIL is a single utility program that allows you to do the same functions as MARKINIT, MARKDUMP, MARKREST, and MARKCON. These utilities must be executed in separate job steps; MARKUTIL is executed in a single job step.

MARKUTIL is executed once and you are able to initialize, dump, restore, and condense the common library. Also within the same job step, you can transfer cataloged items from one common library to another. The utility control (UC) statement is the only allowable input statement available with MARKUTIL.

Multiple UC statements can be used that specify the operation that you want to perform, such as INIT, DUMP, and REST. Within the UC statement, a common library filename/ddname specification allows you to choose the common library that you want the operation to be performed upon. This lets you execute multiple operations within one job step. For clarity, certain operation types are shown as separate job steps in this section. This is followed by an example of multiple UC statements used in one job step. See the [VISION:Builder Reference Summary](#) for details on the UC statement.

MARKUTIL Initialization

The common library must be initialized before definitions or procedures and requests are cataloged. The MARKUTIL program and a UC statement with an operation code of INIT are used to initialize the common library. Normally, this program is executed once to allocate the required space and format the area used for the common library. Therefore, inadvertent re-execution of the INIT operation on the same data set name results in erasing items previously cataloged.

You can initialize either a BDAM or VSAM common library, depending upon the access method selected when your COMLIB was installed. A VSAM common library requires the DEFINE CLUSTER function of the IDCAMS utility.

[Figure 12-1](#) shows the JCL for defining a VSAM cluster for the common library.

[Figure 12-2 on page 12-8](#) shows the JCL statements to initialize a BDAM and VSAM common library, respectively.

Notes

```

//define JOB (accounting information)
//JOB CAT DD DSN=vsam.catalog,DISP=SHR
//step1 EXEC PGM=IDCAMS,REGION=1536K
//SYS PRINT DD SYSOUT=a
//SYS IN DD *
DEFINE CLUSTER -
1 (NAME (common.library) -
2 VOLUME (xxxxxx) -
NUMBERED -
3 RECORDSIZE (507 507) -
SHAREOPTIONS (3 3) -
4 CISZ (4096) -
TRK (1 1) -
5 DATA (NAME (common.library.DATA))
/*
//

```

Figure 12-1 Define Cluster for a VSAM Common Library in OS/390

Notes	Explanation
1	This statement identifies the common library (file-ID) for which the cluster is to be defined. Substitute the name of your common library in place of 'common.library.'
2	This statement indicates the disk volume where the common library is to reside. Substitute the volume serial number in place of 'xxxxxx.'
3	This statement specifies the minimum record size allowable for the common library. (See additional information following these notes.)
4	This statement specifies the control interval that provides efficient processing in most cases.
5	This statement defines the data portion of the common library cluster. Substitute the name of your common library in place of 'common.library.'

The directory blocking factor (number of directory detail records in a DDR) is determined for VSAM common libraries by the record size specified in the cluster definition. The record size must be at least 507-bytes, but can be larger.

A 507-byte record size means that all records on the common library are 507-bytes. A 507-byte DDR allows 15 directory entries (each entry is 32-bytes). Twelve of these directory entries point to data records or items. A larger record size increases the directory blocking factor; each additional 32-bytes increases the directory entries by 1. The greater the blocking factor, the fewer accesses there are to locate an item on the common library.

The best record size is the one that is closest to the average size of a common library item and fits without excessive waste in the chosen control interval size (CI size). You should also keep in mind the directory blocking factor the record size allows.

It is advisable to choose a CI size that best balances a block size which is adequately large and data transfer time that does not stall the processing throughput. CI sizes of 2048 (2K) and 4096 (4K) are good CI sizes. VSAM is likely to compute a CI size of 2048 if not overridden in the cluster.

With a record size of 507 and CI size of 2048, four records are stored in the CI. Each CI contains VSAM control information; a 4-byte CIDF (one per CI) and a 3-byte RDF (one per record). Thus, four records in the CI equals 2028-bytes (4 times 507), plus 16-bytes of VSAM control fields (4 for the CIDF and 12 for the RDFs). Therefore, with the 2048 CI size and 507 record size, 2044-bytes of the CI are used and 4-bytes unused.

The CI size and record size can best be decided after discovering the average size of your common library items which will vary from one installation to another. The average can be determined from a dump of an existing common library. Directory records are easily spotted and bytes 1 through 4 of each directory detail entry contains the binary length of the data item. For a VSAM common library, the length of a data item can exceed both record and CI length. Optimally, however, the record size and longest data item should not exceed the CI size.

Use share options of (3 3) unless you intend to access the common library cluster with software other than VISION:Builder, such as IDCAMS REPO; if so, use (2 3).

Notes

BDAM

```

//          JOB
//JOBLIB   DD DSN=builder.loadlib,DISP=SHR
1 //init   EXEC PGM=MARKUTIL,REGION=1536K
2 //M4LIST DD SYSOUT=a
3 //M4LIB  DD DSN=common.library,DISP=(NEW,CATLG,DELETE),
//          UNIT=sysda,SPACE=(TRK,n,,CONTIG)
4 //M4WORK DD DUMMY
//M4INPUT DD *
5          UCINIT
//
//

```

VSAM

```

//          JOB
//JOBLIB   DD DSN=builder.loadlib,DISP=SHR
//JOB CAT  DD DSN=vsam.catalog,DISP=SHR
1 //init   EXEC PGM=MARKUTIL,REGION=1536K
2 //M4LIST DD SYSOUT=a
3 //M4LIB  DD DSN=common.library,DISP=SHR
4 //M4WORK DD DUMMY
//M4INPUT DD *
5          UCINIT
//
//

```

Figure 12-2 BDAM/VSAM Common Library Initialization Using MARKUTIL in OS/390

Notes	Explanation
1	Execute the MARKUTIL service program.
2	The M4LIST statement defines the location of the system output device for this job.
3	M4LIB represents the common library you are initializing in this run. Note that the space must be contiguous for BDAM format common libraries; if it is not, the common library is initialized to the smaller contiguous size and the non-contiguous area is ignored.
4	An M4WORK file is required for all service program operations. Since it is not actually used by the INIT operation, it can be dummied.
5	Enter UC in positions [9-10] with the operation code INIT in positions [11-14] to initialize the common library.

MARKUTIL Dump and Restore

MARKUTIL can be used to backup and restore a common library. A UC statement with an operation code of DUMP copies the contents of the common library to the sequential M4WORK data set. A UC statement with a REST operation code copies the contents of M4WORK to a common library.

When the DUMP function is used, all items in the common library that have gone beyond their expiration date are flagged for deletion. The output to M4LIST is a listing of names and types of entries that are dumped and the items that are flagged. Upon execution of the REST function, the flagged items may or may not be restored to the common library, depending on a specification entered on the UC statement.

[Figure 12-3](#) illustrates the JCL to dump a BDAM and VSAM common library. [Figure 12-4 on page 12-11](#) illustrates the JCL to restore a BDAM and VSAM common library.

Notes	BDAM
	<pre> // JOB //JOBLIB DD DSN=builder.loadlib,DISP=SHR 1 //dump EXEC PGM=MARKUTIL,REGION=1536K 2 //M4LIST DD SYSOUT=a 3 //M4LIB DD DSN=common.library,DISP=SHR 4 //M4WORK DD DSN=work.file,DISP=(NEW,CATLG,DELETE), // UNIT=sysda,SPACE=(TRK,(n,n),RLSE) //M4INPUT DD * 5 UCDUMP // // </pre>
	VSAM
	<pre> // JOB //JOBLIB DD DSN=builder.loadlib,DISP=SHR //JOB CAT DD DSN=vsam.catalog,DISP=SHR 1 //dump EXEC PGM=MARKUTIL,REGION=1536K 2 //M4LIST DD SYSOUT=a 3 //M4LIB DD DSN=common.library,DISP=SHR 4 //M4WORK DD DSN=work.file,DISP=(NEW,CATLG,DELETE), // UNIT=sysda,SPACE=(TRK,(n,n),RLSE) //M4INPUT DD * 5 UCDUMP // // </pre>

Figure 12-3 BDAM/VSAM Common Library Dump Using MARKUTIL in OS/390

Notes	Explanation
1	Execute the MARKUTIL service program.
2	The M4LIST statement defines the location of the system output device for this job.
3	M4LIB represents the common library you are dumping in this run.
4	This statement represents the sequential data set onto which the common library is dumped. This data set must be large enough to contain the common library.
5	Enter UC in positions [9-10] with the operation code DUMP in positions [11-14] to unload the common library.

Notes**BDAM**

```

//          JOB
//JOBLIB   DD DSN=builder.loadlib,DISP=SHR
1 //rest   EXEC PGM=MARKUTIL,REGION=1536K
2 //M4LIST  DD SYSOUT=a
3 //M4LIB   DD DSN=common.library,DISP=SHR
4 //M4WORK  DD DSN=work.file,DISP=SHR
//M4INPUT DD *
5          UCREST
//
/*
//

```

VSAM

```

//          JOB
//JOBLIB   DD DSN=builder.loadlib,DISP=SHR
//JOB CAT  DD DSN=vsam.catalog,DISP=SHR
1 //rest   EXEC PGM=MARKUTIL,REGION=1536K
2 //M4LIST  DD SYSOUT=a
3 //M4LIB   DD DSN=common.library,DISP=SHR
4 //M4WORK  DD DSN=work.file,DISP=SHR
//M4INPUT DD *
5          UCREST
//
/*
//

```

Figure 12-4 BDAM/VSAM Common Library Restore Using MARKUTIL in OS/390

Notes	Explanation
1	Execute the MARKUTIL service program.
2	The M4LIST statement defines the location of the system output device for this job.
3	M4LIB represents the common library that receives the backup sequential data set that is restored in common library format.
4	This statement specifies the sequential backup data set used to restore to M4LIB. It must have been created with the DUMP option.
5	Enter UC in positions [9-10] with the operation code REST in positions [11-14] to restore your common library.

MARKUTIL Condense

The MARKUTIL service program can be used to free the unused area in the common library. A UC statement is used with the COND operation code specified to condense the common library by making available the area from which items were previously deleted.

The COND function dumps the usable items from the common library to a sequential data set (M4WORK), frees the common library of the “deleted” items, and restores the contents of the sequential data set to the common library.

[Figure 12-5](#) illustrates the OS/390 JCL statements used for condensing a BDAM or VSAM common library.

Notes	BDAM
	<pre> // JOB //JOBLIB DD DSN=builder.loadlib,DISP=SHR 1 //cond EXEC PGM=MARKUTIL,REGION=1536K 2 //M4LIST DD SYSOUT=a 3 //M4LIB DD DSN=common.library,DISP=SHR 4 //M4WORK DD DSN=work.file,DISP=(NEW,DELETE), // UNIT=sysda,SPACE=(TRK,(n,n),RLSE) 5 //M4INPUT DD * UCCOND // // </pre>
	VSAM
	<pre> // JOB //JOBLIB DD DSN=builder.loadlib,DISP=SHR //JOB CAT DD DSN=vsam.catalog,DISP=SHR 1 //cond EXEC PGM=MARKUTIL,REGION=1536K 2 //M4LIST DD SYSOUT=a 3 //M4LIB DD DSN=common.library,DISP=SHR 4 //M4WORK DD DSN=work.file,DISP=(NEW,DELETE), // UNIT=sysda,SPACE=(TRK,(n,n),RLSE) 5 //M4INPUT DD * UCCOND // // </pre>

Figure 12-5 Condensing a BDAM/VSAM Common Library Using MARKUTIL in OS/390

Notes	Explanation
1	Execute the MARKUTIL service program.
2	The M4LIST statement defines the location of the system output device for this job.
3	M4LIB represents the common library to be condensed in this run.
4	This statement represents the sequential data set onto which the common library is to be dumped and from which it is to be restored. This data set must be large enough to contain your common library.
5	Enter UC in positions [9-10] with the operation code COND in positions [11-14] to condense your common library.

MARKUTIL Copy and Merge

The MARKUTIL service program can be used to copy or merge cataloged items from one or more common libraries (sending) to another common library (receiving).

The receiving common library is M4LIB. The sending common library or common libraries can be M4LIB1 through M4LIB9. The sending common library is determined by specifying the DTF/DDname of the common library, on the UC statement.

The COPY and MERG functions are implemented by the utility control (UC) statement. The COPY function copies cataloged items from the sending common library or common libraries (M4LIBn) to the receiving common library (M4LIB). The MERG function merges the entire contents of the sending common library or common libraries (M4LIBn) to the receiving common library (M4LIB).

Duplicate names of cataloged items can occur when dealing with multiple common libraries. You may want to copy a particular version to the receiving common library. This is accomplished by specifying the sending common library on the UC statement. If a sending common library is not specified, the default is M4LIB1; the same is true of the MERG function. Therefore, M4LIB1 must be specified in your JCL if you do not specify a sending common library on the UC statement.

You can replace an existing cataloged item on the receiving common library by specifying a replace option on the UC statement. You may also specify that the replacing item be the newest or oldest version. If you do not specify the newest or oldest version, the first occurrence is copied.

If the replace option is not used, items cannot be copied/merged to the receiving common library if they already exist on that common library. If this is the case, the operation is suppressed, a message is output to M4LIST and the run continues with the next UC statement. Any copy or merge operation causes the output of an informational message to M4LIST, indicating the success or failure of the operation.

Note: Copy operations with REPLACE RULES referencing cataloged definitions which do not have ITEM TRACKING dates will fail and issue a message.

Figure 12-6 illustrates OS/390 JCL for MARKUTIL COPY and MERG functions.

```

Notes          BDAM

              //          JOB
1             //JOBLIB DD DSN=builder.loadlib,DISP=SHR
2             //copy  EXEC PGM=MARKUTIL,REGION=1536K
3             //M4LIST DD SYSOUT=a
4             //M4LIB  DD DSN=common.library,DISP=SHR
5             //M4LIB1 DD DSN=altlib1,DISP=SHR
              //M4LIB5 DD DSN=altlib5,DISP=SHR
6             //M4WORK DD DUMMY
7             //M4INPUT DD *
              UCCOPY  FDfilename          YNEW
              UCMEGR          M4LIB5

              /*
              //

              VSAM

              //          JOB
1             //JOBLIB DD DSN=builder.loadlib,DISP=SHR
2             //JOB CAT DD DSN=vsam.catalog,DISP=SHR
3             //copy  EXEC PGM=MARKUTIL,REGION=1536K
4             //M4LIST DD SYSOUT=a
5             //M4LIB  DD DSN=common.library,DISP=SHR
6             //M4LIB1 DD DSN=altlib1,DISP=SHR
7             //M4LIB5 DD DSN=altlib5,DISP=SHR
              //M4WORK DD DUMMY
              //M4INPUT DD *
              UCCOPY  FDfilename          YNEW
              UCMEGR          M4LIB5

              /*
              //
    
```

Figure 12-6 Copy and Merge BDAM/VSAM Common Libraries Using MARKUTIL in OS/390

Notes	Explanation
1	Execute the MARKUTIL service program.
2	The M4LIST statement defines the location of the system output device for this run.
3	M4LIB represents the common library that cataloged items are copied into.
4	M4LIB1 represents the common library that cataloged items are copied from. This is the default sending common library; see notes 6 and 7.

Notes	Explanation
5	An M4WORK file is required for all MARKUTIL operations. Since it is not used by the COPY operation, it can be dummied.
6	Enter UC in positions [9-10] with the operation code COPY in positions [11-14] to copy cataloged items from one common library to another. The item(s) to be copied are specified in the item name entry on the UC statement(s). The named file definition is to be replaced into M4LIB, as long as the version in the sending common library is more recently updated than the version in M4LIB. Since the sending common library is not specified on the UC statement, the default sending common library is M4LIB1.
7	The entire contents of M4LIB5 is merged into the receiving common library (M4LIB).

MARKUTIL and Multiple UC Statements

This section contains examples of the MARKUTIL service program when multiple UC statements are used, specifying different operation codes. [Figure 12-7](#) illustrates the OS/390 JCL.

Notes

```

//          JOB
//JOBLIB   DD DSN=builder.loadlib,DISP=SHR
1 //multuc EXEC PGM=MARKUTIL,REGION=1536K
2 //M4LIST  DD SYSOUT=A
3 //M4LIB   DD DSN=common.library,DISP=(NEW,CATLG),
//          UNIT=sysda,SPACE=(TRK,n,,CONTIG)
4 //M4LIB1  DD DSN=alt.lib1,DISP=OLD
5 //M4LIB2  DD DSN=alt.lib2,DISP=OLD
6 //M4LIB5  DD DSN=alt.lib5,DISP=OLD
7 //M4LIB9  DD DSN=alt.lib9,DISP=OLD
8 //M4WORK  DD DSN=work.file,
//          DISP=(NEW,CATLG,DELETE),
//          UNIT=sysda,SPACE=(TRK,(n,n),RLSE)
9 //M4INPUT DD *
//          UCINIT
//          UCCOND                M4LIB2
//          UCMERG                M4LIB9
//          UCCOPY FDfilename     M4LIB5
//          UCCOPY TDtran         mastfile
//
/*
//

```

Figure 12-7 MARKUTIL and Multiple UC Statements in One OS/390 Job Step

Notes	Explanation
1	Execute the MARKUTIL service program.
2	The M4LIST statement defines the location of the system output device for this run.
3	M4LIB specifies the common library that is to be initialized.
4	This statement specifies the default data set that the second UCCOPY statement uses since a ddname is not coded on the UC statement. The items listed on the UC statement are to be copied from M4LIB1 to M4LIB.
5	This statement specifies the common library to be condensed with the UCCOND statement.
6	This statement specifies the common library that cataloged items are to be copied from to M4LIB, with the first UCCOPY statement.
7	This statement specifies the common library to be merged with M4LIB, as a result of the UCMERG statement.
8	This statement specifies the sequential data set that is used in the UCCOND statement that condenses M4LIB2.
9	The UC statements are performed in the order in which they are coded, in this example: INIT, COND, MERG, COPY, COPY.

MARKINIT Common Library Initialization

The MARKINIT utility program can be used instead of MARKUTIL to initialize your common library. Normally, MARKINIT is executed once to allocate the required space and format the area used for the common library. Therefore, the inadvertent re-execution of MARKINIT on the same data set name results in erasing items previously cataloged.

[Figure 12-8](#) shows the OS/390 JCL statements to invoke MARKINIT for a BDAM and VSAM common library, respectively. Execute a DEFINE CLUSTER before you initialize a VSAM common library. See the DEFINE CLUSTER JCL in [Figure 12-1 on page 12-7](#).

```

Notes
//          JOB (accounting information)
//JOBLIB   DD DSN=builder.loadlib,DISP=SHR
1 //step1  EXEC PGM=MARKINIT,REGION=1536K
2 //M4LIB  DD DSN=common.library,DISP=(NEW,CATLG,DELETE),
//          UNIT=sysda,SPACE=(TRK,n,,CONTIG)
3 //M4LIST DD SYSOUT=a
// *
//

VSAM
//          JOB (accounting information)
//JOBLIB   DD DSN=builder.loadlib,DISP=SHR
//JOBCAT  DD DSN=vsam.catalog,DISP=SHR
1 //step1  EXEC PGM=MARKINIT,REGION=1536K
2 //M4LIB  DD DSN=common.library,DISP=OLD
3 //M4LIST DD SYSOUT=a
// *
//
    
```

Figure 12-8 BDAM/VSAM Initialization Using MARKINIT in OS/390

Notes	Explanation
1	Execute the MARKINIT utility program.
2	The common library to be initialized.
3	The system output device for this run.

MARKDUMP and MARKREST Common Library Dump and Restore

MARKDUMP and MARKREST can be used instead of MARKUTIL to backup or restore a common library or to copy a common library from one area to another. MARKDUMP copies the contents of the common library to the sequential work data set.

[Figure 12-9](#) and [Figure 12-10](#) illustrate the OS/390 JCL for MARKDUMP and MARKREST.

For a VSAM common library, the following steps are required:

- The common library is dumped.
- The cluster for the common library is deleted.
- A new larger cluster is defined (in cases where more disk space is needed for the common library).
- The copy of the old common library is restored to the new common library.

MARKREST includes an initialization function (for VSAM common libraries only); therefore, it is not necessary to initialize a new common library when using MARKREST.

For a BDAM common library, you use MARKINIT to initialize a new common library prior to executing MARKREST.

Notes

```

//          JOB   (accounting information)
//JOBLIB   DD DSN=builder.loadlib,DISP=SHR
//*  STEP 1 DUMP THE LIBRARY TO THE WORK DATA SET
1 //step1 EXEC PGM=MARKDUMP,REGION=1536K
//M4LIST  DD SYSOUT=a
2 //M4LIB   DD DSN=common.library,DISP=SHR
3 //M4WORK  DD DSN=work.file,DISP=(NEW,PASS,DELETE),
//          UNIT=sysda,SPACE=(TRK,(n,n),RLSE)
//*  STEP 2 INITIALIZE A NEW LARGER LIBRARY
4 //step2 EXEC PGM=MARKINIT,REGION=1536K
//M4LIST  DD SYSOUT=a
//M4LIB   DD DSN=new.common.library,
//          DISP=(NEW,CATLG,DELETE),
//          UNIT=sysda,SPACE=(CYL,n,,CONTIG)
//*  STEP 3 RESTORE THE LIBRARY FROM THE WORK DATA SET
5 //step3 EXEC PGM=MARKREST,REGION=1536k
//M4LIST  DD SYSOUT=a
6 //M4LIB   DD DSN=new.common.library,DISP=SHR
7 //M4WORK  DD DSN=work.file,DISP=(OLD,DELETE)
//
//

```

Figure 12-9 Dump and Restore a BDAM Common Library Using MARKDUMP, MARKINIT, and MARKREST in OS/390

Notes	Explanation
1	Execute the MARKDUMP utility program.
2	The common library to be dumped.
3	The sequential data set onto which the common library is to be dumped. This data set must be large enough to contain the common library.
4	Execute the MARKINIT utility program.

Notes	Explanation
5	Execute the MARKREST utility program.
6	The common library to which the contents of the work data set are to be restored.
7	The sequential data set that was created by the dump program and from which the common library is restored.

Notes

```

//          JOB (accounting information)
//JOBLIB   DD DSN=builder.loadlib,DISP=SHR
//JOB CAT  DD DSN=vsam.catalog,DISP=SHR
//* STEP 1  DUMP THE LIBRARY TO THE WORK DATA SET
1 //step1  EXEC PGM=MARKDUMP,REGION=1536K
2 //M4LIB  DD DSN=common.library,DISP=SHR
3 //M4WORK DD DSN=work.file,DISP=(NEW,CATLG,DELETE),
//        UNIT=sysda,SPACE=(TRK,(n,n),RLSE)
//M4LIST  DD SYSOUT=a
//*
4 //* STEP 2  PURGE THE OLD VSAM LIBRARY
//*        AND DEFINE CLUSTER FOR A NEW, LARGER LIBRARY
//step2   EXEC DSN=IDCAMS,REGION=1536K
//SYS PRINT DD SYSOUT=A
//SYS IN  DD *
DELETE common.library -
PURGE
5 DEFINE CLUSTER -
(NAME(new.common.library) -
VOLUME(xxxxxx) -
NUMBERED -
SHAREOPTIONS(3 3) -
CISZ(4096)
RECORDSIZE(507 507) -
CYL(1 1)) -
DATA(NAME(new.common.library.DATA))
//*
6 //* STEP 3  RESTORE THE LIBRARY FROM THE WORK DATA SET
//step3   EXEC PGM=MARKREST,REGION=1536K
//M4LIST  DD SYSOUT=a
7 //M4LIB  DD DSN=new.common.library,DISP=SHR
8 //M4WORK DD DSN=work.file,DISP=SHR
//*
//

```

Figure 12-10 Dump and Restore a VSAM Common Library Using MARKDUMP, MARKINIT, and MARKREST in OS/390

Notes	Explanation
1	Execute the MARKDUMP utility program.
2	The common library to be dumped.
3	The sequential data set onto which the common library is to be dumped. This data set must be large enough to contain the common library.

Notes	Explanation
4	Step 2 purges the old common library and allocates a new, larger common library.
5	The DEFINE for the new, larger common library. (See MARKUTIL Initialization on page 12-6 for additional information about the common library cluster definition.)
6	Execute the MARKREST utility program.
7	The common library to which the contents of the work data set is to be restored.
8	The sequential data set created by the dump program and from which the common library is restored.

MARKCON Common Library Condense

The MARKCON utility program can be used in place of MARKUTIL to free the unused area in the common library. MARKCON condenses the common library by making available the area from which items were previously deleted.

MARKCON dumps the usable items from the common library to a sequential work data set, frees the common library of the “deleted” items, and restores the contents of the sequential data set to the common library.

[Figure 12-11](#) illustrates OS/390 JCL statements used for condensing a BDAM or VSAM common library.

```

Notes
      //          JOB   (accounting information)
      //JOBLIB    DD   DSN=builder.loadlib,DISP=SHR
1  //step1      EXEC  PGM=MARKCON,REGION=1536K
2  //M4LIB      DD   DSN=common.library,DISP=SHR
3  //M4WORK     DD   DSN=work.file,DISP=(NEW,DELETE),
      //          UNIT=sysda,SPACE=(TRK,(n,n),RLSE)
      //M4LIST   DD   SYSOUT=a
      // *
      //

      //          JOB   (accounting information)
      //JOBLIB    DD   DSN=builder.loadlib,DISP=SHR
      //JOB CAT   DD   DSN=vsam.catalog,DISP=SHR
1  //STEP1      EXEC  PGM=MARKCON,REGION=1536K
2  //M4LIB      DD   DSN=common.library,DISP=SHR
3  //M4WORK     DD   DSN=work.file,DISP=(NEW,DELETE),
      //          UNIT=sysda,SPACE=(TRK,(n,n),RLSE)
      //M4LIST   DD   SYSOUT=a
      // *
      //

```

Figure 12-11 Condensing a BDAM/VSAM Common Library Using MARKCON in OS/390

Notes	Explanation
1	Execute the MARKCON utility program.
2	The common library to be condensed.
3	The sequential work data set onto which the common library is to be dumped and from which it is restored. This data set must be large enough to contain the common library.

Using Cataloged Procedures and Requests

Cataloged procedures and requests are VISION:Builder procedures and requests that are saved in a common library for use during application processing runs.

Cataloged Procedure and Request Maintenance

Catalog maintenance operations are used to save, replace, insert, delete, list, and dump cataloged procedures and requests that are in your common library. Procedures and requests can be cataloged and maintained individually or in groups.

All procedure and request catalog maintenance operations are performed in application processing runs. See [Chapter 2, VISION:Builder Runs, Run Control, and Execution JCL](#) for a description of the JCL for an application processing run.

The various catalog maintenance operations are specified on the CATALOG statement. The CATALOG statement is included in the application processing run following the CONTROL statements. You can specify a scan/terminate processing run (keyword SCAN on the CONTROL statement) if you want to perform catalog maintenance without executing your application. See the [ASL Reference Guide](#) for the details on the operations and entries available in the CATALOG statement.

Cataloged Procedure and Request Execution

Cataloged procedures and requests are executed in application processing runs. The procedures and requests to be executed (individual and/or groups) are retrieved from the common library by including an INCLUDE statement in your application following the run control statements. See the [ASL Reference Guide](#) for a description of the INCLUDE statement.

Multiple INCLUDE statements can be used and interleaved with other instream procedures and requests in the application. The order of the INCLUDE statements will dictate the order that the cataloged procedures and requests will appear within the application.

Listing and Retrieving Common Library Items

The catalog listing and retrieval functions of VISION:Builder allow the user to list and retrieve items previously cataloged and stored in the common library.

Common Library Listing Operations

The various operations for listing the contents of the common library are specified in the LISTLIB statement. See the [VISION:Builder ASL Reference Guide](#) for the details on the common library listing operations that are available with the LISTLIB statement.

There are listing operations that allow you to list the names of the various items stored in the common library by type of item. You can also get a list of everything in the common library by name within type. You can obtain glossary listings for the various items in the common library which document their contents in detail.

The common library listing operations are performed in a definition/maintenance run. See [Chapter 2, VISION:Builder Runs, Run Control, and Execution JCL](#) for a description of the JCL required to execute a definition/maintenance run.

Common Library Retrieval Operations

The various operations for retrieving the contents of the common library are specified in the RETRIEVE statement. See the [VISION:Builder ASL Reference Guide](#) for the details on the common library retrieval operations that are available with the RETRIEVE statement.

The retrieval operations allow you to retrieve the various items from your common library in source statement format. The common library retrieval operations are performed in a source statement retrieval (SSR) definition/maintenance run. Source statement retrieval runs are a distinct type of run, indicated by the presence of a RETRIEVE statement.

The SSR capability allows you to retrieve the following in source statement form:

- File definitions
- Run data group definitions
- Array and table definitions
- Transaction group definitions
- Cataloged procedures and requests

Retrieved source statements are output to the file indicated by the ddname M4SSOUT. The input files to a source statement retrieval run are M4LIB and M4INPUT; output files are M4LIST and M4SSOUT. No other files can be included.

In the SSR run, the run control group consists of one CONTROL statement and one or more RETRIEVE statements.

M4SSOUT is a file containing 80-character, fixed length records. If you specify a RETRIEVE EOF statement, the record output to M4SSOUT is a /* statement. The VISION:Builder own code hooks 10, 20, 21, 63, and 91 are supported under SSR runs.

Sample Source Statement Retrieval Runs

This section contains sample job control for retrieving source statements from the common library with VISION:Builder.

[Figure 12-12](#) shows the interactions that take place in a source statement retrieval run. [Figure 12-13](#) shows the OS/390 JCL for a source statement retrieval run.

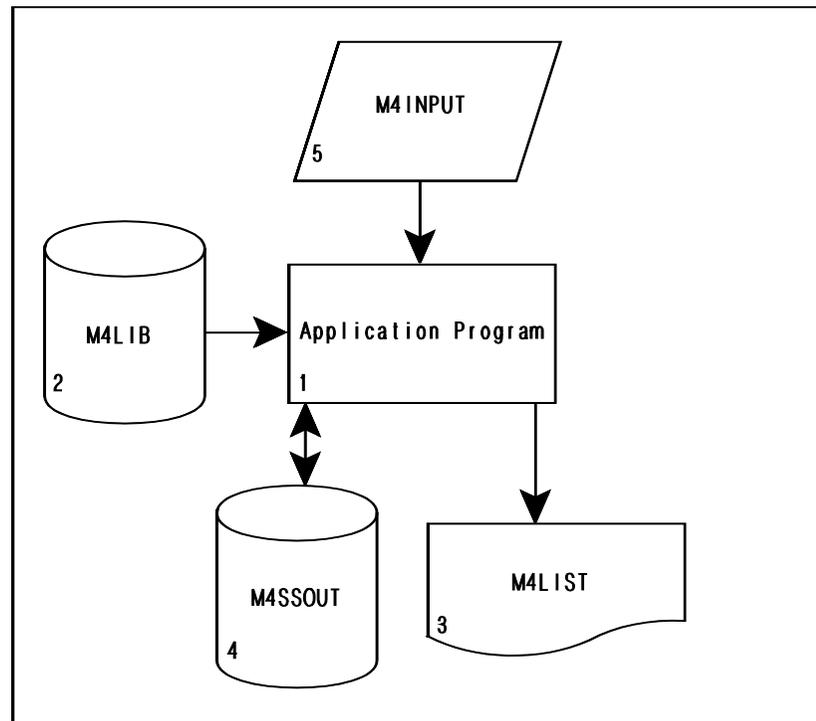


Figure 12-12 Interactions of Source Statement Retrieval Run

Notes

```

//          JOB (accounting information)
/* JCL FOR SOURCE STATEMENT RETRIEVAL RUN **
//JOBLIB   DD DSN=builder.loadlib,DISP=SHR
1 //step    EXEC PGM=MARKIV,REGION=1536K
//M4LIB    DD DSN=common.library,DISP=SHR
3 //M4LIST  DD SYSOUT=a
4 //M4SSOUT DD DSN=ssout.file,DISP=(NEW,CATLG,DELETE),
//          SPACE=(TRK,(n,n),RLSE),
//          UNIT=sysda
5 //M4INPUT DD *
CONTROL
RETRIEVE GROUP, ITEM . . .
.
.
/*
//

```

Figure 12-13 OS/390 JCL for a Source Statement Retrieval Run

Notes	Explanation
1	Execute MARKIV for a source statement retrieval run.
2	The M4LIB represents the common library from which items are to be retrieved.
3	The M4LIST statement defines the location of the system output device for this run. The output includes a listing of the source statements and diagnostic messages.
4	This statement defines the file (M4SSOUT) used to hold the source statements that are retrieved.
5	The M4INPUT defines the data set containing source statement retrieval statements. The CONTROL statement must be the first of these statements. One or more RETRIEVE statements defining the retrieval operation(s) immediately follow the CONTROL statement.

Fixed Syntax Statement Usage Considerations

When fixed syntax statements are used instead of ASL statements for an SSR run, the following considerations will apply:

Except for RP and CT statements, all other M4INPUT records of any type (for example, JCL or SYSTEM commands) which follow the RC statement are copied directly to the source statement output file (M4SSOUT); no other actions are performed on them. CT statements and continuation CT statements containing retrieval operators are not copied to the source statement output file (M4SSOUT).

Statements containing retrieval operators are only used to define items to be retrieved and output in source statement form. They are processed before any subsequent input stream statements are accessed.

Because all statements, except run control group and the CT statements that specify SSR operations, are copied from M4INPUT directly to M4SSOUT, you can create a data set on M4SSOUT consisting of common library source statements, JCL and data, which can be used in a later run.

When a CT statement with an SSR operation is encountered, the specified source statements are retrieved from the common library and output to the M4SSOUT before any subsequent M4INPUT statements are accessed.

VISION:Builder HTML Document Style Customization

One of the Alternate Report Output Methods in VISION:Builder is an HTML document. VISION:Builder uses templates containing HTML code to define the layout and appearance of the report content, when it is displayed as a document by a web browser. A default template is delivered with VISION:Builder to use in generating HTML documents. These templates include HTML style specifications such as background color, text color, font, alignment, and so on, that the browser uses when displaying the content of the document. Therefore, a template representing a specific appearance is known as a style.

You can customize the default style delivered with VISION:Builder as you prefer, and create up to 99 other styles to use for different reports prepared as HTML documents. You can specify the style number for a report using one of the following methods:

- Setting it in the Report Method window in VISION:Workbench for DOS
- Including the STYLE keyword on the FORMAT command.

If left blank, the default style is used. At execution time, the HTML templates for each style used in an application must be present in a library. This library is identified with the M4HTBASE ddname.

The template for each HTML style consists of 5 related members in the M4HTBASE library. Each template member name for a style ends in a unique 2-digit suffix. The 00 suffix represents the default style. Up to 99 sets of template members with a suffix of 01 through 99 can be created to define additional styles as needed. The 5 members making up the template for a specific style must all end with the same suffix. The following table describes the purpose of each of the 5 members defining a specific style. Note that the HTPRIXnn and HTPRIYnn

members serve the same basic function, and either one or the other is used for each report. The HTLTOCnn member is only used when the HTPRIXnn member is chosen for a report.

M4HTBASE Library	
Member Name	Contains the HTML code that...
HTPRIXnn	Defines the frames used for an HTML document that includes Level 1 Subtitle specifications.
HTPRIYnn	Defines the frames used for an HTML document that does not include any Level 1 Subtitle specifications.
HTBODYnn	Describes the Body frame for the document. This frame contains the report data in a layout similar to a printed page. The body frame contains all subtitle lines, detail lines, and summary lines specified for the report content.
HTHEADnn	Describes the Heading frame for the document. This frame contains the Page Title and Column Heading lines for the report, similar to a printed page.
HTLTOCnn	Describes the Left-Hand-Table-of-Contents frame for the document. This frame contains the Level 1 Subtitle data for the report and can be used as a hyperlink index to the corresponding section of detail and/or summary data lines related to the subtitle data value.

Within the HTBODYnn, HTHEADnn, and HTLTOCnn members is a section of HTML code delimited by the <STYLE> and </STYLE> tags. It is this section of HTML code that can be modified to create additional styles. To create a new style, copy the members from the default or existing user style and modify the HTML code within the style section, to provide the appropriate appearance of the new style. You must not modify, insert, or remove any HTML code outside of the style section of these members.

Detail and summary lines in the Body frame as well as Page Title and Column Heading lines in the Heading frame are inserted into the appropriate frames as pre-formatted text bracketed by the <PRE> and </PRE> tags. Subtitle lines in the Body frame are bracketed by the <H5> and </H5> tags. Level 1 Subtitle data in the Left-Hand-Table-of-Contents frame are bracketed by the <A> and tags. The style section for each frame contains style specifications for each tag. Changing the style parameters for a tag changes the appearance of the related report data when it displays using the browser.

The Heading frame for the default FRAMESET definition (in members HTPRIX00 and HTPRIY00) contains the specification of SCROLLING=NO. To enable scrolling for this frame, in the HTPRIXnn and HTPRIYnn members change the specification to SCROLLING=YES. If you are familiar with HTML coding, you can modify or extend the FRAMESET and FRAME specifications in these two members to meet specific needs or preferences. The portion of the screen space

reserved for each FRAMESET is a parameter that you can change to provide styles that can better meet the needs for a particular report. You can also define an additional frame to include a corporate logo or other static information to be displayed with each report.

When a VISION:Builder report is prepared as an HTML document, the report content is written out as either 3 or 4 members into the partitioned dataset specified as the output destination. The following table shows the correspondence of the style members in the M4HTBASE library to the output member in the destination dataset. Note that the \$MAIN output member is copied from either the HTPRIXnn or HTPRIYnn template member. The LTOC output member is present only when the HTPRIXnn template member is used.

M4HTBASE Library Member Name	Action	Output Member Name
HTPRIXnn	Copied as is when Level 1 Subtitle specifications are present.	\$MAIN
HTPRIYnn	Copied as is when Level 1 Subtitle specifications are not present.	\$MAIN
HTHEADnn	Page Title and Column Heading lines are merged into this member at the "....." placeholder location.	HEAD
HTBODYnn	Subtitle, detail, and summary lines are merged into this member at the "....." placeholder location.	BDYnnnnn
HTLTOCnn	Level 1 subtitle fields are merged into this member at the "....." placeholder location.	LTOC

The following HTML code represents each of the default style members provided by Computer Associates.

Member HTPRIX00

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<TITLE>VISION:Builder Report Type X, Default Style</TITLE>
</HEAD>
<FRAMESET COLS="19%,*">
  <FRAME NAME="ltoc" SRC="ltoc.html" MARGINHEIGHT=0 MARGINWIDTH=0>
  <FRAMESET ROWS="13%,*" BORDER=1>
    <FRAME NAME="head" SRC="head.html" MARGINHEIGHT=2 MARGINWIDTH=2
      SCROLLING=NO>
    <FRAME NAME="body" SRC="body.html" MARGINHEIGHT=2 MARGINWIDTH=2>
  </FRAMESET>
</FRAMESET>
</HTML>
```

Member HTPRIY00

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<TITLE>VISION:Builder Report Type Y, Default Style</TITLE>
</HEAD>
<FRAMESET ROWS="13%,*" BORDER=1>
  <FRAME NAME="head" SRC="head.html" MARGINHEIGHT=2 MARGINWIDTH=2
    SCROLLING=NO>
  <FRAME NAME="body" SRC="body.html" MARGINHEIGHT=2 MARGINWIDTH=2>
</FRAMESET>
</HTML>
```

Member HTHEAD00

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<TITLE>VISION:Builder Report Heading Frame, Default Style</TITLE>
<STYLE>
<!--
BODY {color:black; background:aqua}
PRE {color:black; background:aqua; font-size:x-small}
-->
</STYLE>
</HEAD>
<BODY>
<PRE>
..... <Place Holder - Do Not Remove or Reposition>
</PRE>
</BODY>
</HTML>
```

Member HTBODY00

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<TITLE>VISION:Builder Report Detail Frame, Default Style</TITLE>
<STYLE>
<!--
BODY {color:blue; background:white}
H5 {color:black; background:lightgrey; font-size:x-small;
  font-weight:bold; font-family:courier;
  text-align:left; line-height:normal}
PRE {color:blue; background:white; font-size:x-small}
-->
</STYLE>
</HEAD>
<BODY><PRE>
..... <Place Holder - Do Not Remove or Reposition>
</BODY>
</HTML>
```

Member HTLTOC00

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<BASEFONT SIZE=2>
<HEAD>
<TITLE>VISION:Builder Report Table of Contents Frame,
      Default Style</TITLE>
<STYLE>
<!--
BODY {color:black; background:skyblue}
H4   {color:yellow; background:blue; font-size:medium; font-weight:bold}
A    {color:maroon; background:skyblue; font-size:x-small}
-->
</STYLE>
</HEAD>
<BODY>
<H4>Report Contents</H4>
<UL>
.....    <Place Holder - Do Not Remove or Reposition>
</UL>
</BODY>
</HTML>
```


Index

A

access methods, 10-9
alternate list file, 2-26
alternate report files, 2-18, 2-19, 2-20, 2-21
alternate report output methods
 single-step JCL, 2-59
 three-step JCL, 2-63
application, 1-3
 development overview, 1-3
 run, 1-6
 run control group, 2-5

B

batch free-form input, 8-1
batch query language, 7-1, 7-2
 definition step, 7-6
 execution, 7-1
 glossary, 7-5
 processing, 7-5
 report step, 7-6
 sample execution, 7-6, 7-7
 sort step, 7-6
blocking factors ISAM, 10-9

C

call attach facility, 11-1, 11-3
calling VISION:Builder, 10-3

cataloged procedure and request maintenance, 12-21
cataloged procedures and request, 12-21
checkpoint/restart, 6-1, 6-5, 6-8, 6-14
 CHKP flag, 6-13
 CKPTID, 6-10
 CKPTID flag, 6-13, 6-14
 examples, 6-5, 6-7
 timing, 6-3
 with DB2, 6-14
comma-delimited output, 2-61, 2-66
common library utilities, 12-1
configuration requirements, 1-1
CP statement, 6-1
CP statement (checkpoint), 2-7
CT statement, 12-21

D

DB statement, 2-6
DB2, 11-1
 IMS/ESA checkpoints, 6-14
DD statements
 JCL statements, 2-9
defining data structures, 1-5
definition/maintenance run, 1-5, 2-8
 JCL requirements, 2-8
definitions
 stored in the common library, 12-2

DL/I, 3-4, 3-10

file definition, DBD and PSB relationship, 3-2

interface and retrieval, 3-1

JCL requirements, 3-7

MARKDLI, DBDs, and PSBs, 3-1

non-unique IMS key/search field, 3-10

secondary indexing, 3-10

F

free-form input, 8-1

G

GSAM DBD, 6-9

GSAM PCB, 6-9

GSAM PSB, 6-10

H

HTML

code, 13-3

output, 2-59, 2-61, 2-62, 2-63, 2-66, 2-67

style, 13-1

templates, 2-59, 2-62, 2-63, 2-66

I

IMS attach facility, 11-1, 11-2

IMS/ESA checkpoint/restart, 6-14

initialization program formats the common lib, 12-6

introduction, 1-1

IT statement (item tracking), 2-7

J

JCL for VISION:Builder runs, 2-1

application runs, 2-9

application runs alternate report files, 2-18, 2-19, 2-20

application runs alternate report output methods, 2-59, 2-63

application runs coordinated files, 2-22

application runs master files, 2-22

application runs report summary files, 2-44, 2-45, 2-47, 2-48

application runs single-step processing, 2-9, 2-10, 2-15

application runs three-step processing, 2-11, 2-13, 2-14

application runs update a master file, 2-29

application runs update-in-place, 2-48

concatenation of input data sets, 10-3

definition/maintenance run, 2-8

override parameters, 10-1

program analyzer, 2-34, 2-35, 2-36, 2-39, 2-40, 2-41

scan runs, 2-33

Job Control Language (JCL)

for common library utilities, 12-6

for source statement retrieval, 12-23

L

LB statement, 2-7

limiting records read during input processing, 10-7

linking to VISION:Builder, 10-3

listing common library items, 12-22

M

maintenance

to definitions, 12-21

utilities, 12-6

MARKDLI, 3-1, 3-7, 3-8, 6-1

MARKDLIX, 6-1, 6-9

master and coordinated files, 2-22, 2-23, 2-24, 2-25

message processing, 10-8

monitoring activity

multiple common libraries, 12-2

O

OC statement (own-code), 2-7

online execution of VISION:Builder, 9-1

CLISTs, 9-4

free-form input, 9-5, 9-6, 9-10, 9-11

messages, 9-4

OLX commands, 9-4

query language, 9-5

sample run, 9-11, 9-12

security interfaces, 9-5

optimization of resources, 10-7, 10-8

own-code

dynamic integration, 5-2

facilities, 5-1

flow, 5-7

hook 10, 5-2, 5-3, 5-9, 5-10, 5-18, 5-21

hook 11, 5-3, 5-10

hook 20, 5-11, 5-12, 5-18

hook 21, 5-9, 5-12, 5-13, 5-18

hook 30, 5-7, 5-14

hook 50, 5-15, 5-20

hook 51, 5-3, 5-16

hook 60, 5-16

hook 61, 5-3, 5-17

hook 62, 5-17

hook 63, 5-9, 5-12, 5-13, 5-17, 5-18

hook 70, 5-15, 5-20

hook 91, 5-9, 5-10, 5-20, 5-21

hook 92, 5-21

hook 93, 5-22

hook descriptions, 5-9, 5-10, 5-11, 5-12, 5-14, 5-15, 5-16, 5-17, 5-18, 5-20, 5-21, 5-22

hook naming conventions, 5-5

interrupts, 5-4

linkage, 5-4

obtaining space, 5-3

static integration, 5-2

user I/O, 5-1, 5-9, 5-10, 5-23, 5-28, 5-37

user I/O, COBOL example, 5-37

variable length fields, 5-23

P

PA statement (program analyzer), 2-7

PAL

program analyzer, 2-36

PARM information, 10-4

plain text output, 2-62, 2-67

processing data

application run, 1-6

program analyzer

JCL requirements, 2-34, 2-39, 2-40

Q

query language

batch query language or online execution, 7-1

R

RA statement (arrays), 2-7

raw data output, 2-62, 2-67

RC statement (run control), 2-6

relational support, 11-1

report file optimization, 10-6

report manager, 2-50

report summary files, 2-44, 2-45, 2-47, 2-48

JCL requirements, 2-42, 2-43, 2-44, 2-45, 2-47

REPTSIZE, 2-32

resource optimization, 10-7, 10-8

restart

checkpoint/restart, 6-1

retrieving

common library items, 12-22

source statements, 12-22

RF statement (files), 2-7

RG statement (transaction groups), 2-7

RP statement (run parameters), 2-7

RT statement (relational tables), 2-7

run control group, 2-5

S

sample reports, 2-32, 2-33, 2-34
scan runs
 JCL requirements, 2-32, 2-33, 2-34
shared common libraries, 12-2
single-step processing, 1-6
 JCL requirements, 2-9, 2-16
SORTSIZE, 2-32
source statement retrieval, 12-22
storage optimization, 10-8
subfile and alternate list
 JCL requirements, 2-26, 2-28

T

tab-delimited output, 2-61, 2-66
three-step processing, 1-6
 JCL requirements, 2-11, 2-13, 2-14
TSO attach facility, 11-1, 11-2

U

UC statement, 12-6
update master file
 JCL requirements, 2-29, 2-30
update-in-place (no-sort), 2-48, 2-49, 2-50
user I/O
 own-code, 5-1
using the common library, 12-1
utilities to maintain the common library, 12-1

V

variable-spanned records, 10-9
VSAM, 4-1
 alternate index as a user file, 4-3
 alternate index paths, 4-1, 4-2, 4-3, 4-4
 generic and duplicate keys, 4-4
 record support, 4-1

W

WH statement, 2-7