

FOCUS for S/390

VSAM Write Data Adapter User's Manual

Contents

1	Introduction	1-1
	Operating Environment Considerations	1-1
2	Describing VSAM Data Sources to FOCUS.....	2-1
	Simple ISAM and Key-Sequenced VSAM Data Sources	2-1
	Attributes Used to Describe VSAM Data Sources	2-2
	Describing Groups.....	2-2
	Complex and Key-Sequenced VSAM Data Sources	2-5
	Describing Positionally-Related Records.....	2-5
	Describing Data Sources With Unrelated Records.....	2-7
	VSAM Repeating Groups With RECTYPES	2-9
	Describing VSAM Repeating Groups Using MAPFIELD.....	2-11
	Alternate Indexes.....	2-15
	Advanced Topics.....	2-17
	Combining RECTYPE and OCCURS Segments	2-17
	Reading Complex Data Sources With User-Written Procedures	2-20
3	Maintaining VSAM Data Sources.....	3-1
	The IDCAMS Utility Program	3-2
	Accessing VSAM Data Sources	3-2
	Simultaneous Usage (SU) Mode for VSAM Data Sources in MVS.....	3-3
	Starting VSAM/SU Source Machines	3-3
	Querying the VSAM/SU FOCUS Database Server.....	3-4
	Modifying Records: The MATCH and NEXT Statements.....	3-4
	Accessing Fields in Joined Data Structures With LOOKUP.....	3-6
	Modifying Multiple Data Sources: The COMBINE Facility	3-7
	Accessing Fields in Combined Data Sources With the FIND Function.....	3-7
	Controlling the Current Record Position With REPOSITION	3-8
	Looping Through Files With REPEAT	3-9
	Modifying Variable-Length OCCURS Segments	3-11
	Modifying Fixed-Length OCCURS Segments.....	3-15
	Modifying Data Sources With Unrelated Records	3-18

Contents

A	Debugging Techniques	A-1
B	User Exits for Non-FOCUS Data Sources	B-1
	The Dynamic and Re-Entrant GNTINT Private User Exit	B-1
	Functional Requirements.....	B-2
	Implementation.....	B-3
	User-coded Data Access Modules	B-8
	Re-Entrant VSAM Compression Exit: ZCOMP1.....	B-10
	Linking ZCOMP1	B-10
	What Happens When You Use ZCOMP1	B-10
	ZCOMP1 Parameter List.....	B-11
	Index	I-1

Due to the nature of this material, this document refers to numerous hardware and software products by their trade names. In most, if not all cases, these designations are claimed as trademarks or registered trademarks by their respective companies. It is not the publisher's intent to use any of these names generically. The reader is therefore cautioned to investigate all claimed trademark rights before using any of these names other than to refer to the product described.

Information Builders, the Information Builders logo, Web390, and FOCUS are registered trademarks of Information Builders, Inc.

CICS, DB2, IBM, RACF, VM/ESA, and VTAM are registered trademarks, and C/MVS, COBOL/370, IMS, MVS, MVS/XA, MVS/ESA, and SQL/DS are trademarks of International Business Machines corporation.

Teradata is a registered trademark of NCR International, Inc.

Oracle is a registered trademark of the Oracle Corporation.

ADABAS is a registered trademark of Software A.G.

Millennium is a registered trademark of Geac Computer Systems, Inc.

Model 204 is a registered trademark of Computer Corporation of America.

CA-IDMS/DB, CA-DATACOM/DB, CA-ACF2, and CA-TOP SECRET are registered trademarks of Computer Associates, Inc.

Nomad is a registered trademark of Aonix.

Copyright © 2000 by Information Builders, Inc. All rights reserved. This manual or parts thereof, may not be reproduced in any form without the written permission of Information Builders, Inc.

Printed in the U.S.A

Preface

This manual describes how to update VSAM data sources using FOCUS and is intended for users working in the OS/390, MVS, CMS or VM/ESA operating environments. In the OS/390 and MVS environment, the Simultaneous Usage mode for VSAM (VSAM/SU) is used to enable several people to maintain a VSAM file concurrently using FOCUS MODIFY procedures.

This document assumes familiarity with VSAM data sources and with standard FOCUS MODIFY procedures.

How This Manual Is Organized

This manual includes the following chapters:

Chapter/Appendix		Contents
1	<i>Introduction</i>	Brief description of VSAM Write Data Adapter usage and operating requirements
2	<i>Describing VSAM Data Sources to FOCUS</i>	Tells how to describe VSAM data sources to FOCUS and how they relate to FOCUS data sources
3	<i>Maintaining VSAM Data Sources</i>	Tells how to use FOCUS to create new VSAM data sources and update existing ones. Discusses use of MODIFY and MAINTAIN features with VSAM structures.
A	<i>Debugging Techniques</i>	Describes where to start when debugging FOCUS update routines for VSAM data structures
B	<i>User Exits for Non-FOCUS Data Sources</i>	Covers GNTINT user exit, user-coded data-access modules and the ZCOMP1 VSAM data compression exit

Documentation Conventions

The following conventions apply throughout this manual:

UPPERCASE	Denotes a command that you must enter in uppercase, exactly as shown.
<i>lowercase italic</i>	Denotes a value that you must supply.
<u>underscore</u>	Underscore indicates a default option.
Punctuation	Required as shown.

{ }	Indicates two choices from which you must choose one. You type one of these choices, not the braces.
[]	Indicates optional parameters. None of them is required, but you may select one of them. Type only the information within the brackets, not the brackets.
	Separates two mutually exclusive choices in a syntax line. Type one of these choices, not the symbol.
...	Indicates that you can enter a parameter multiple times. Type only the information, not the ellipsis points.
. . .	Indicates that there are (or could be) intervening or additional commands.

Related Publications

Generally, standard FOCUS attributes apply when describing VSAM data sources (see your FOCUS documentation).

See the *Information Builders Technical Publications Catalog* for the most up-to-date listing and prices of technical publications, plus ordering information. To obtain a catalog, contact the Publications Order Department at (800) 969-4636.

You can also visit our World Wide Web site, <http://www.informationbuilders.com>, to view a current listing of Information Builders publications or to place an order.

Customer Support

Do you have questions about the VSAM Write Data Adapter?

Call Information Builders Customer Support Service (CSS) at (800) 736-6130 or (212) 736-6130. Customer Support Consultants are available Monday through Friday between 8:00 a.m. and 8:00 p.m. EST to address your VSAM update questions. Information Builders consultants can also give you general guidance regarding product capabilities and documentation. Please be ready to provide your six-digit site code number (xxxx.xx) when you call.

You can also access support services electronically, 24 hours a day, with InfoResponse Online. InfoResponse Online is accessible through our World Wide Web site, <http://www.informationbuilders.com>. It connects you to the tracking system and known-problem database at the Information Builders support center. Registered users can open, update, and view the status of cases in the tracking system and read descriptions of reported software issues. New users can register immediately for this service. The technical support section of www.informationbuilders.com also provides usage techniques, diagnostic tips, and answers to frequently asked questions.

To learn about the full range of available support services, ask your Information Builders representative about InfoResponse Online, or call (800) 969-INFO.

Information You Should Have

To help our consultants answer your questions most effectively, be ready to provide the following information when you call:

- Your six-digit site code number (*xxxx.xx*).
- The FOCEXEC procedure (preferably with line numbers).
- Master File with picture (provided by CHECK FILE).
- Run sheet (beginning at login, including call to FOCUS), containing the following information:
 - ? RELEASE
 - ? FDT
 - ? LET
 - ? LOAD
 - ? COMBINE
 - ? JOIN
 - ? DEFINE
 - ? STAT
 - ? SET/? SET GRAPH
 - ? USE
 - ? TSO DDNAME OR CMS FILEDEF

Preface

- The exact nature of the problem:
 - Are the results or the format incorrect; are the text or calculations missing or misplaced?
 - The error message and code, if applicable.
 - Is this related to any other problem?
- Has the procedure or query ever worked in its present form? Has it been changed recently? How often does the problem occur?
- What release of the operating system are you using? Has it, FOCUS, your security system, or an interface system changed?
- Is this problem reproducible? If so, how?
- Have you tried to reproduce your problem in the simplest form possible? For example, if you are having problems joining two data sources, have you tried executing a query containing just the code to access the data source?
- Do you have a trace file?
- How is the problem affecting your business? Is it halting development or production? Do you just have questions about functionality or documentation?

User Feedback

In an effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual. Please use the Reader Comments form at the end of this manual to relay suggestions for improving the publication or to alert us to corrections. You can also use the Document Enhancement Request Form on our Web site, <http://www.informationbuilders.com>.

Thank you, in advance, for your comments.

Information Builders Consulting and Training

Interested in training? Information Builders Education Department offers a wide variety of training courses for this and other Information Builders products.

For information on course descriptions, locations, and dates, or to register for classes, visit our World Wide Web site (<http://www.informationbuilders.com>) or call (800) 969-INFO to speak to an Education Representative.

1 Introduction

A free data adapter for reading VSAM data structures comes with the standard FOCUS report writer. To update VSAM data structures with FOCUS, however, you will also need the VSAM Write Data Adapter, as well as either the FOCUS MODIFY or MAINTAIN option.

Existing VSAM Master Files can be used as is, or you can write new VSAM Master Files following the instructions provided in Chapter 2, *Describing VSAM Data Sources to FOCUS*. While all FOCUS read, write, and update features are usable with VSAM, some work slightly differently due to the nature of the VSAM data structures and those differences are addressed in this manual.

The VSAM Write Data Adapter only supports VSAM key-sequenced data sets (KSDS) and entry-sequenced data sets (ESDS). FOCUS does not support relative-record data sets (RRDS) VSAM files. To retrieve data from RRDS VSAM files, you must write your own access routines using SUFFIX=PRIVATE. See Appendix B for information on writing custom data-access routines.

Operating Environment Considerations

The VSAM Write Data Adapter works with FOCUS in the VM/ESA and OS/390 operating environments.

Under VM/ESA VSAM data sources on DOS-formatted minidisks are all available to the data adapter. When VSAM is installed using the Cross Machine Interface (XMI), however, only read access is supported and VSAM data cannot be modified.

Under OS/390, users have the choice of submitting jobs in batch mode or running them interactively and can access files controlled by either the Integrated Catalog Facility (ICF) or the VSAM Master Catalog.

2 Describing VSAM Data Sources to FOCUS

Topics:

- Simple ISAM and Key-Sequenced VSAM Data Sources
- Complex and Key-Sequenced VSAM Data Sources
- Advanced Topics

This chapter tells how to describe VSAM data sources and how they relate to FOCUS data sources. To access a VSAM data source, you must first describe it using FOCUS terminology. VSAM data source descriptions are stored as FOCUS Master Files (MASTERS). The FOCUS programs used to read or update them are called FOCEXECs.

Generally, standard FOCUS attributes apply for describing VSAM data sources as well (see your FOCUS documentation); exceptions are noted in this chapter. Additional concepts in processing VSAM data sources require special FOCUS terminology at both the field and segment levels. This chapter discusses specific aspects of creating FOCUS Master Files unique to VSAM sources. ESDS files are described in the same manner as KSDS files, but lack group keys.

Sites that wish to convert existing COBOL VSAM file descriptions directly into FOCUS Master Files can do so with the COBOL FD Translator, an optional FOCUS product that is sold and installed separately.

Simple ISAM and Key-Sequenced VSAM Data Sources

Most techniques for describing fixed-format records described in the FOCUS documentation assume sequential QSAM or entry-sequenced VSAM files without keys. A new element is introduced with VSAM, the key or group key. Group keys consist of one or more fields identifying the various record types in the file. In FOCUS representations of such files, each unique record type is assigned its own segment.

Note: For VSAM data sources, you must allocate (in OS/390) or DLBL (in DOS/VSE and VM/ESA) the Master File name to the data source's CLUSTER component.

Attributes Used to Describe VSAM Data Sources

Standard FOCUS attributes for describing files, segments, and fields apply to VSAM data sources, with the following conditions.

File Suffixes for VSAM Data Sources

The SUFFIX attribute in a VSAM data source declaration must have a value of ISAM or VSAM.

Segment Name for VSAM Data Sources

The SEGNAME for the first or root segment of a VSAM data source must be ROOT. Remaining segments can have any valid name. The only exception is files with unrelated RECTYPES, for which the first SEGNAME must be DUMMY.

All non-repeating data goes in the root segment. The remaining segments may be given any valid 1- to 8-character name.

Every segment except the root is a descendant, or child, of another segment. The PARENT attribute supplies the segment name of the hierarchical parent or owner of the current segment. If no PARENT attribute appears, the default is the immediate preceding segment. PARENT names may be one to eight characters.

SEGTYPE for VSAM Data Sources

The SEGTYPE attribute must be S0 for VSAM data sources.

Describing Groups

Keys of VSAM data sources are defined in SEGMENT declarations as GROUPs consisting of one or more fields. The GROUP declaration has the following syntax

Syntax

How to Code GROUP Declarations

`GROUP=keyname , ALIAS=KEY , USAGE=Ann , ACTUAL=Ann , $`

where:

keyname

Can have up to 66 characters.

Single-segment VSAM structures may only have one key field but you must still describe that with a GROUP declaration. Each group must have ALIAS=KEY to create a mapping to the offset and length of the VSAM key.

Groups can be assigned to provide convenient names for referencing groups of fields. If, for instance, you had three fields for employee: last name; first name; and social security number and used them frequently to identify the employee. You could identify them in your Master File as a GROUP named EMPINFO to reference them as a unit. When using the GROUP feature for non-key fields, the parameter ALIAS= must still be used, but should not be named KEY.

For group fields you must supply both USAGE and ACTUAL formats for alphanumeric fields. Their format lengths must exactly equal the sum of all subordinate field lengths.

If the fields making up a group key are not alphanumeric fields, the format of the group key is still alphanumeric, but its length is determined differently. The ACTUAL length still equals the sum of the subordinate field lengths. The USAGE format, however, is the sum of the internal storage lengths of the subordinate fields.

Determine the internal storage lengths as follows:

- Type I fields have a value of 4.
- Type F fields have a value of 4.
- Fields of 8 bytes have a USAGE of P15, P16, P17.n (sign and decimal for a total of 15 digits). Fields of 16 bytes will have a USAGE of P17 or larger.
- Type D fields have a value of 8.
- Alphanumeric fields have field length values equal to the number of characters they contain.

Example Describing a Group Field

Consider the following example:

```
GROUP=A      ALIAS=KEY ,USAGE=A14      ,ACTUAL=A8      , $
FIELDNAME=F1 ,ALIAS=F1  ,USAGE=P6      ,ACTUAL=P2      , $
FIELDNAME=F2 ,ALIAS=F2  ,USAGE=I9      ,ACTUAL=I4      , $
FIELDNAME=F3 ,ALIAS=F3  ,USAGE=A2      ,ACTUAL=A2      , $
```

The lengths of the ACTUAL attributes for subordinate fields F1, F2, and F3 total 8, which is the length of the ACTUAL attribute of the group key. The lengths of the USAGE attributes for the subordinate fields total 17. However, the length of the group key USAGE attribute is found by adding their internal storage lengths as specified by their field types: 8 for USAGE=P6, 4 for USAGE=I9, and 2 for USAGE=A2, for a total of 14.

The GROUP declaration USAGE attribute tells FOCUS how many positions to use for the group key. If a Master File does not completely describe the full key at least once, FOCUS returns the following warning message:

```
(FOC1016) INVALID KEY DESCRIPTION IN MASTER FILE
```

When you expand on the key in a RECTYPE file, describe the key length in full on the last non-OCCURS segment on each data path. Do not describe a group with ALIAS=KEY for OCCURS segments.

Describing VSAM Data Sources to FOCUS

Note: Since all group fields must be defined in alphanumeric format, those that include numeric component fields should not be used as verb objects in a report request.

Using a library example, the first field, PUBNO, could be described as a group key. The publisher's number consists of three elements, a number that identifies the publisher, one that identifies the author, and one that identifies the title. Until now they have been described as a single, ten-digit number made up of these elements. They could have been described as a group key, consisting of a separate field for each element if the file were a VSAM data structure. The FOCUS description of this would look as follows:

```
FILE=LIBRARY5, SUFFIX=VSAM,$
SEGMENT=ROOT, SEGTYPE=S0,$
GROUP=BOOKKEY, ALIAS=KEY, USAGE=A10, ACTUAL=A10, $
FIELDNAME=PUBNO, ALIAS=PN, USAGE=A3, ACTUAL=A3, $
FIELDNAME=AUTHNO, ALIAS=AN, USAGE=A3, ACTUAL=A3, $
FIELDNAME=TITLNO, ALIAS=TN, USAGE=A4, ACTUAL=A4, $
FIELDNAME=AUTHOR, ALIAS=AT, USAGE=A25, ACTUAL=A25, $
FIELDNAME=TITLE, ALIAS=TL, USAGE=A50, ACTUAL=A50, $
FIELDNAME=BINDING, ALIAS=BI, USAGE=A1, ACTUAL=A1, $
FIELDNAME=PRICE, ALIAS=PR, USAGE=D8.2N, ACTUAL=D8, $
FIELDNAME=SERIAL, ALIAS=SN, USAGE=A15, ACTUAL=A15, $
FIELDNAME=SYNOPSIS, ALIAS=SY, USAGE=A150, ACTUAL=A150, $
FIELDNAME=RECTYPE, ALIAS=B, USAGE=A1, ACTUAL=A1, $
```

When using group fields with multiple formats in queries, you must account for each position, including trailing blanks and leading zeros. The following example illustrates how to access a group field with multiple formats in a query.

```
GROUP=GRPGB, ALIAS=KEY, USAGE=A8, ACTUAL=A8, $
FIELDNAME=FIELD1, ALIAS=F1, USAGE=A2, ACTUAL=A2, $
FIELDNAME=FIELD2, ALIAS=F2, USAGE=I8, ACTUAL=I4, $
FIELDNAME=FIELD3, ALIAS=F3, USAGE=A2, ACTUAL=A2, $
```

The values in fields F1 and F3 may include some trailing blanks, and values in field F2 may include leading zeros. When issuing a query for the group, you must account for each position:

```
IF GRPB EQ 'A 0334BB'
```

This introduces the possibility of positional errors, but you can eliminate that possibility by using slashes (/) to separate the components of group keys:

```
IF GRPB EQ 'A/334/BB'
```

FOCUS assumes blanks and leading zeros as needed to fill out the key.

Complex and Key-Sequenced VSAM Data Sources

Like complex sequential files, complex VSAM data sources could consist of positionally or non-positionally related records. This section tells how to describe both kinds of data structures to FOCUS, beginning with those containing positionally-related records.

Describing Positionally-Related Records

Some VSAM data sources are structured so that descendant records relate to each other through concatenating key fields. That is, the key field(s) of a parent record serves as the first part of the key of a child record. In such cases, the segment's key fields must be described to FOCUS using a GROUP declaration. Each segment's GROUP key fields will consist of the renamed key fields from the parent segment plus the unique key field from the child record.

Consider the following VSAM data structure with three types of records. The ROOT records have a key consisting of the publisher's number, PUBNO. The BOOKINFO segment has a key consisting of that same publisher's number, plus a hard or soft-cover indicator, BINDING. The SERIANO segment key consists of the first two elements, plus a record type field, RECTYPE.

The Master File for this structure looks as follows:

```

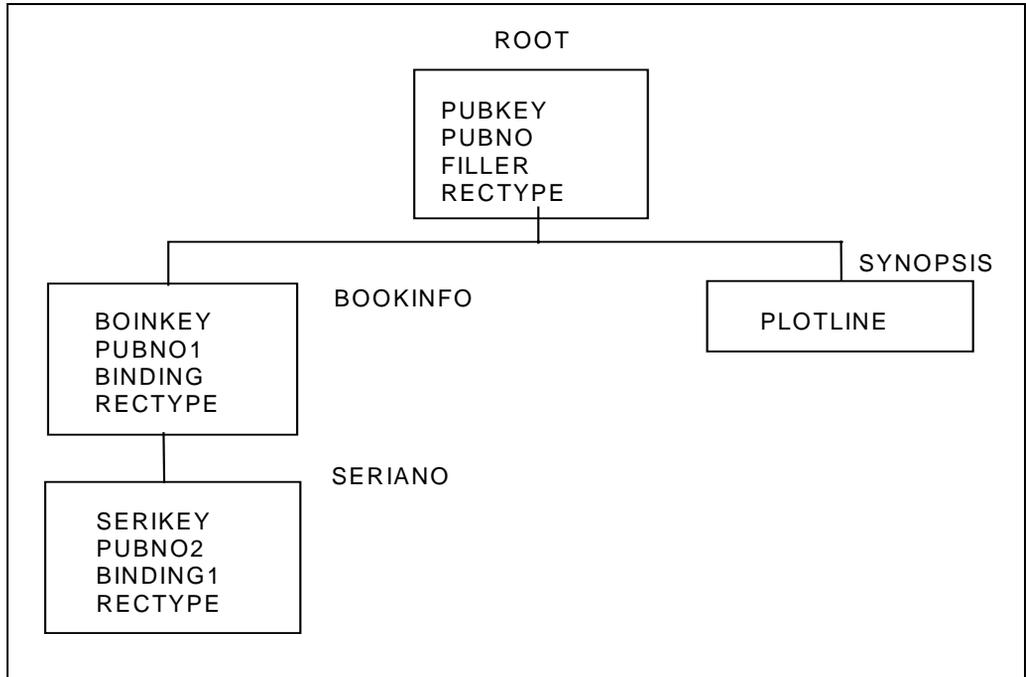
FILENAME=LIBRARY6, SUFFIX=VSAM,$
SEGNAME=ROOT, SEGTYPE=S0,$
GROUP=PUBKEY, ALIAS=KEY, USAGE=A10, ACTUAL=A10, $
FIELDNAME=PUBNO, ALIAS=PN, USAGE=A10, ACTUAL=A10, $
FIELDNAME=FILLER, ALIAS=, USAGE=A1, ACTUAL=A1, $
FIELDNAME=RECTYPE, ALIAS=1, USAGE=A1, ACTUAL=A1, $
FIELDNAME=AUTHOR, ALIAS=AT, USAGE=A25, ACTUAL=A25, $
FIELDNAME=TITLE, ALIAS=TL, USAGE=A50, ACTUAL=A50, $
SEGNAME=BOOKINFO, PARENT=ROOT, SEGTYPE=S0,$
GROUP=BOINKEY, ALIAS=KEY, USAGE=A11, ACTUAL=A11, $
FIELDNAME=PUBNO1, ALIAS=P1, USAGE=A10, ACTUAL=A10, $
FIELDNAME=BINDING, ALIAS=BI, USAGE=A1, ACTUAL=A1, $
FIELDNAME=RECTYPE, ALIAS=2, USAGE=A1, ACTUAL=A1, $
FIELDNAME=PRICE, ALIAS=PR, USAGE=D8.2N, ACTUAL=D8, $
SEGNAME=SERIANO, PARENT=BOOKINFO, SEGTYPE=S0,$
GROUP=SERIKEY, ALIAS=KEY, USAGE=A12, ACTUAL=A12, $
FIELDNAME=PUBNO2, ALIAS=P2, USAGE=A10, ACTUAL=A10, $
FIELDNAME=BINDING1, ALIAS=B1, USAGE=A1, ACTUAL=A1, $
FIELDNAME=RECTYPE, ALIAS=3, USAGE=A1, ACTUAL=A1, $
FIELDNAME=SERIAL, ALIAS=SN, USAGE=A15, ACTUAL=A15, $
SEGNAME=SYNOPSIS, PARENT=ROOT, SEGTYPE=S0, OCCURS=VARIABLE,$
FIELDNAME=PLOTLINE, ALIAS=PLOTL, USAGE=A10, ACTUAL=A10, $

```

Describing VSAM Data Sources to FOCUS

Note that the length of the key fields specified in the USAGE and ACTUAL attributes of a GROUP declaration is the length of the key fields from the parent segment(s) plus the length of the added field of the child segment. In the example above, the length of the GROUP key SERIKEY equals the length of PUBNO2 and BINDING1, the group key from the parent segment, plus the length of RECTYPE, the field added to the group key in the child segment.

In the sample file, the repetition of the publisher's number as PUBNO1 and PUBNO2 in the descendant segments relates the three types of records. The file can be diagrammed as the following FOCUS structure:



A typical inquiry might request information on price and call numbers by publisher number:

```
PRINT PRICE AND SERIAL BY PUBNO
IF PUBNO EQ 1234567890 OR 9876054321
```

Since PUBNO is part of the key, such a retrieval can be made quickly and processing continues. To further expedite retrieval you could add search criteria based on the BINDING field, which is also part of the key.

The RECTYPE Attribute

Key-sequenced VSAM data sources also use the RECTYPE attribute field to distinguish record types within them. A record type (RECTYPE) field appears in the same position of all records in the file. A parent does not always share its RECTYPE with its descendants. It shares some other identifying piece of information, such as the PUBNO in our example. This field should be included in the parent key, as well as in all of its descendant keys, to relate them.

When using the RECTYPE attribute in VSAM data sources with group keys, the RECTYPE field can only be part of the segment's group key when it belongs to a segment with no descendants, or to a segment whose descendants are described with an OCCURS attribute. In the previous example, the RECTYPE field is added to the group key in the SERIANO segment, the lowest descendant segment in the chain.

If you place the RECTYPE field in the parent segment's portion of the key, the file will be sorted first by record type when FOCUS processes it, destroying the VSAM positional relationship with records in the child segments. In the previous example, if the RECTYPE of the BOOKINFO segment were made part of the group key, all BOOKINFO records would be sorted together and their relationship to the records in SERIANO would be lost.

Describing Data Sources With Unrelated Records

Some VSAM data sources do not have related records. That is, the VSAM key of one record type is independent of the keys of other record types.

For example, consider another VSAM file containing information on our library. This file has three types of records: book information, magazine information, and newspaper information.

There are two possible structures:

- The RECTYPE is the beginning of the key. The key structure is:

RECTYPE B	Book Code
RECTYPE M	Magazine Code
RECTYPE N	Newspaper Code

The sequence of records is:

Book
Book

Magazine
Magazine

Newspaper

Describing VSAM Data Sources to FOCUS

Newspaper

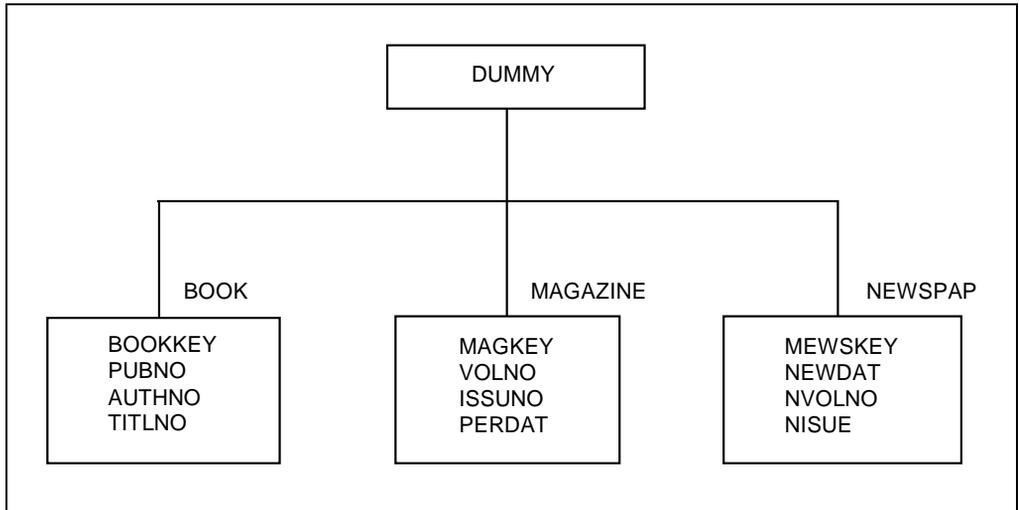
Note the difference between the use of the RECTYPE here and its use in the previous section. In this case, you do want to sort your records by type first (all book information together, all magazine information together, and all newspaper information together), so it is appropriate to include the RECTYPE as part of the key.

- The RECTYPE is not in the beginning of the key or is outside of the key. The key structure is:

Book Code
Magazine Code
Newspaper Code

The sequence of record types can be arbitrary.

Both types of file structure can be represented by the following structure:



Describing a Key and Record Type for a Data Structure With Unrelated Records

To describe either type of file with unrelated records, make the record types descendants of a dummy root segment. The following rules apply:

- The segment name must be DUMMY.
- It must have only one field with an empty name and alias.
- It should not have any keys or record types described.
- The USAGE and ACTUAL formats must be A1.

All other non-OCCURS segments must point to the dummy root as their parent. Except for the dummy segment, all non-OCCURS segments must describe the full VSAM key. If the file does not have a key, the group should not be described. RECTYPES may be anywhere in the record.

The following Master File describes a second, more complex file with unrelated records:

```
FILE=LIBRARY7, SUFFIX=VSAM, $
SEGMENT=DUMMY, $
FIELDNAME=          , ALIAS=          , USAGE=A1          , ACTUAL=A1          , $
SEGMENT=BOOK, PARENT=DUMMY, SEGTYPE=S0, $
GROUP=BOOKKEY       , ALIAS=KEY       , USAGE=A11         , ACTUAL=A11        , $
FIELDNAME=PUBNO     , ALIAS=PN      , USAGE=A3          , ACTUAL=A3         , $
FIELDNAME=AUTHNO    , ALIAS=AN      , USAGE=A3          , ACTUAL=A3         , $
FIELDNAME=TTITLNO   , ALIAS=TN      , USAGE=A4          , ACTUAL=A4         , $
FIELDNAME=RECTYPE   , ALIAS=B       , USAGE=A1          , ACTUAL=A1         , $
FIELDNAME=AUTHOR    , ALIAS=AT      , USAGE=A25         , ACTUAL=A25        , $
FIELDNAME=TITLE     , ALIAS=TL      , USAGE=A50         , ACTUAL=A50        , $
FIELDNAME=BINDING   , ALIAS=BI      , USAGE=A1          , ACTUAL=A1         , $
FIELDNAME=PRICE     , ALIAS=PR      , USAGE=D8.2N      , ACTUAL=D8         , $
FIELDNAME=SERIAL    , ALIAS=SN      , USAGE=A15         , ACTUAL=A15        , $
FIELDNAME=SYNOPSIS , ALIAS=SY      , USAGE=A150        , ACTUAL=A150       , $
SEGMENT=MAGAZINE, PARENT=DUMMY, SEGTYPE=S0, $
GROUP=MAGKEY        , ALIAS=KEY     , USAGE=A11         , ACTUAL=A11        , $
FIELDNAME=VOLNO     , ALIAS=VN      , USAGE=A2          , ACTUAL=A2         , $
FIELDNAME=ISSUNO    , ALIAS=IN      , USAGE=A2          , ACTUAL=A2         , $
FIELDNAME=PERDAT    , ALIAS=DT      , USAGE=A6          , ACTUAL=A6         , $
FIELDNAME=RECTYPE   , ALIAS=M       , USAGE=A1          , ACTUAL=A1         , $
FIELDNAME=PERNAME   , ALIAS=PRN     , USAGE=A50         , ACTUAL=A50        , $
SEGMENT=NEWSPAP, PARENT=DUMMY, SEGTYPE=S0, $
GROUP=NEWSKEY       , ALIAS=KEY     , USAGE=A11         , ACTUAL=A11        , $
FIELDNAME=NEWDAT    , ALIAS=ND      , USAGE=A6          , ACTUAL=A6         , $
FIELDNAME=NVOLNO    , ALIAS=NV      , USAGE=A2          , ACTUAL=A2         , $
FIELDNAME=NISSUE    , ALIAS=NI      , USAGE=A2          , ACTUAL=A2         , $
FIELDNAME=RECTYPE   , ALIAS=N       , USAGE=A1          , ACTUAL=A1         , $
FIELDNAME=NEWNAME   , ALIAS=NN      , USAGE=A50         , ACTUAL=A50        , $
```

VSAM Repeating Groups With RECTYPES

When data sources contain records that have repeating groups, the OCCURS attribute is used to describe a separate segment for the repeating fields.

Note: OCCURS segments are treated as MISSING if they are not activated through an INCLUDE in a FOCUS MODIFY. A period (.) is placed in the first byte of the first occurrence of OCCURS=*n* or OCCURS=*fieldname* to signify that the segment is missing.

In some data sources, however, the repeating fields themselves must be identified according to a RECTYPE indicator.

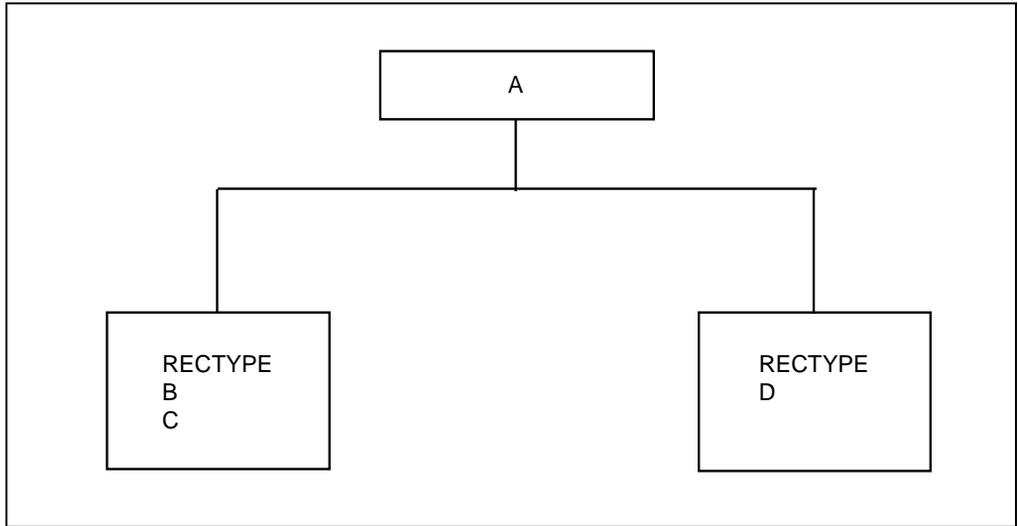
Suppose you want to describe a file that, schematically, looks like this:

A	RECTYPE	B C	RECTYPE	B C
---	---------	-----	---------	-----

Describing VSAM Data Sources to FOCUS

A	RECTYPE	D	RECTYPE	D
---	---------	---	---------	---

You need to describe three segments in your Master File, with A as the root segment, and segments for B, C, and D as two descendant OCCURS segments for A:



Each of the two descendant OCCURS segments in this example depends on the RECTYPE indicator that appears for each occurrence.

All syntax rules for using RECTYPE fields and OCCURS segments also apply to RECTYPES within OCCURS segments.

Since the OCCURS segments are evaluated depending on the contents of the RECTYPE indicator, the RECTYPE must appear at the start of each OCCURS segment. This enables you to describe very complex data sources, including ones with nested and parallel repeating groups that depend on RECTYPES.

In the next example, B/C, and D represent a nested repeating group, and E represents a parallel repeating group.

A	RECTYPE B C	RECTYPE D	RECTYPE E	RECTYPE E
---	-------------	-----------	-----------	-----------

The Master File would be coded as:

```

FILENAME=SAMPLE , SUFFIX=VSAM , $
SEGNAME=ROOT , SEGTYPE=S0 , $
GROUP=GRPKEY      , ALIAS=KEY      , USAGE=A8      , ACTUAL=A8      , $
FIELD=FLD000      , E00      , A08      , A08      , $
FIELD=A_DATA      , E01      , A02      , A02      , $
SEGNAME=SEG001 , PARENT=ROOT , OCCURS=VARIABLE , SEGTYPE=S0      , $
FIELD=RECTYPE    , A01      , A01      , ACCEPT=B OR C , $
FIELD=B_OR_C_DATA , E02      , A08      , A08      , $
SEGNAME=SEG002 , PARENT=SEG001 , OCCURS=VARIABLE , SEGTYPE=S0      , $
FIELD=RECTYPE    , D      , A01      , A01      , $
FIELD=D_DATA     , E03      , A07      , A07      , $
SEGNAME=SEG003 , PARENT=ROOT , OCCURS=VARIABLE , SEGTYPE=S0      , $
FIELD=RECTYPE    , E      , A01      , A01      , $
FIELD=E_DATA     , E04      , A06      , A06      , $
    
```

Describing VSAM Repeating Groups Using MAPFIELD

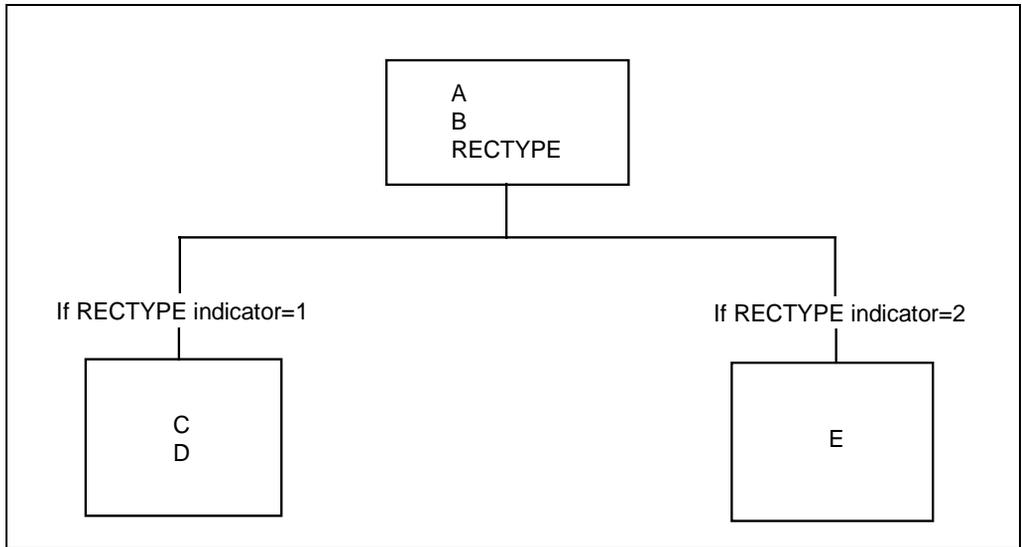
Another possible RECTYPE/OCCURS combination is a record containing a RECTYPE indicator followed by a repeating group. Schematically, such a record would appear like this:

A B	RECTYPE (1)	C D	C D	C D
A B	RECTYPE (2)	E E		

The first record contains header information, values for A and B, followed by an OCCURS segment of C and D that was identified by its preceding RECTYPE indicator. The second record has a different RECTYPE indicator and contains a different repeating group, this time for E.

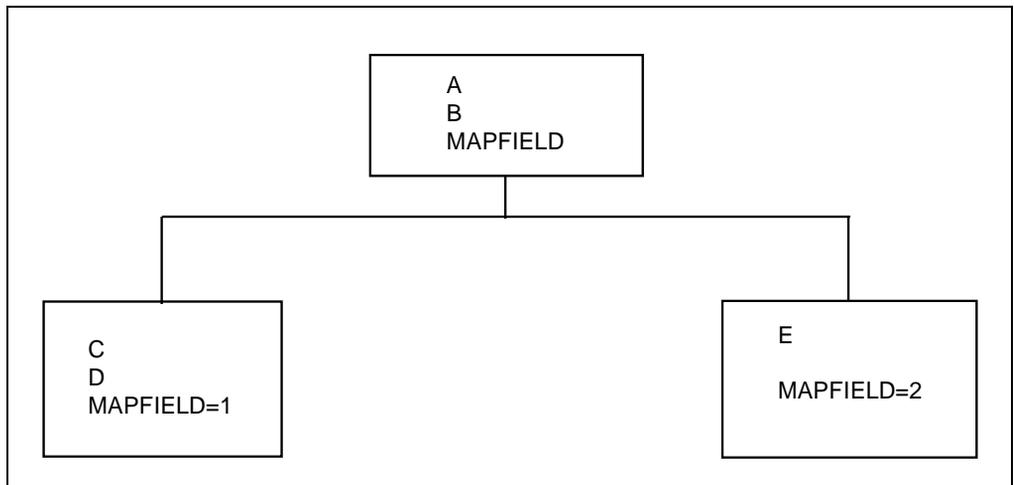
Describing VSAM Data Sources to FOCUS

The following diagram illustrates this relationship.



Since the OCCURS segments are identified by the RECTYPE indicator rather than the parent A/B segment, you can use the keyword MAPFIELD. MAPFIELD identifies a field in the same way RECTYPE does, but since the OCCURS segments will each have their own values for MAPFIELD, the value of MAPFIELD is associated with each OCCURS segment by means of a complementary field named MAPVALUE.

The following diagram illustrates this relationship:



MAPFIELD is assigned as the ALIAS of the field that will be the RECTYPE indicator. You can give this field any name. Otherwise, it is described according to the usual syntax rules.

Syntax**Naming and Defining MAPFIELD Values**

`FIELD=name, ALIAS=MAPFIELD, USAGE=format, ACTUAL=format,$`

where:

`name`

Is the name you provide for this field.

`ALIAS`

MAPFIELD is assigned as the alias of the field that will be the RECTYPE indicator.

`USAGE`

Follows the usual field format.

`ACTUAL`

Follows the usual field format.

Descendant segment values depend on the value of MAPFIELD. They are described as separate segments, one for each possible value of MAPFIELD, and all descend from the segment that has the MAPFIELD. A special field, MAPVALUE, is described as the last field in these descendant segments after the ORDER field, if one was used. The actual MAPFIELD value is supplied as the ALIAS of the MAPVALUE.

Syntax**Defining a MAPFIELD Value Using MAPVALUE**

`FIELD=MAPVALUE, ALIAS=alias, USAGE=format, ACTUAL=format,ACCEPT=list/range ,$`

where:

`MAPVALUE`

Indicates that the segment depends on a MAPFIELD in its parent segment.

`alias`

Is the primary MAPFIELD identifier. If there is an ACCEPT list, this value is any value in the ACCEPT list or range.

`USAGE`

Is the same format as the MAPFIELD format in the parent segment.

`ACTUAL`

Is the same format as the MAPFIELD format in the parent segment.

`list`

Is the list of one or more lines of specified MAPFIELD values for records that have the same segment layout. The maximum number of characters allowed in the list is 255. Each item in the list must be separated by either a blank or the keyword OR. If the list contains embedded blanks or commas, it must be enclosed within single quotation marks (""). The list may contain a single MAPFIELD value. For example:

```
FIELDNAME=MAPFIELD, ALIAS=A, USAGE=A1, ACTUAL=A1,
ACCEPT=A OR B OR C,$
```

`range`

Describing VSAM Data Sources to FOCUS

Is a range of one or more lines of MAPFIELD values for records that have the same segment layout. The maximum number of characters allowed in the range is 255. If the range contains embedded blanks or commas, it must be enclosed in single quotation marks ('). To specify a range of values, include the lowest value, the keyword TO, and the highest value, in that order.

Using the sample file at the beginning of this section, the Master File would look like this:

```
FILENAME=EXAMPLE , SUFFIX=FIX , $
SEGNAME=ROOT , SEGTYPE=S0 , $
FIELD =A      ,           ,A14  ,A14  , $
FIELD =B      ,           ,A10  ,A10  , $
FIELD =FLAG   , MAPFIELD ,A01  ,A01  , $
SEGNAME=SEG001 , PARENT=ROOT , OCCURS=VARIABLE , SEGTYPE=S0 , $
FIELD =C      ,           ,A05  ,A05  , $
FIELD =D      ,           ,A07  ,A07  , $
FIELD =MAPVALUE , 1      , A01  ,A01  , $
SEGNAME=SEG002 , PARENT=ROOT , OCCURS=VARIABLE , SEGTYPE=S0 , $
FIELD =E      ,           ,D12.2 ,D8   , $
FIELD =MAPVALUE , 2      , A01  ,A01  , $
```

Note: MAPFIELD can only exist on an OCCURS segment that has not been positionally re-mapped. The segment definition cannot contain POSITION=fieldname.

VSAM Data and Index Buffers

Two SET commands enable you to establish DATA and INDEX buffers for processing VSAM data sources online.

The AMP sub-parameters BUFND and BUFNI allow MVS BATCH users to enhance the I/O efficiency of their TABLE, TABLEF, MODIFY, and JOIN operations against VSAM data sources by retaining frequently accessed VSAM Control Intervals in memory, rather than on physical DASD. Job throughput is improved by reducing the number of physical I/O operations. These SET commands allow FOCUS users to improve performance in their interactive sessions. In general, BUFND (data buffers) increase the efficiency of physical sequential reads, while BUFNI (index buffers) most benefit JOIN or KEYED access operations.

Syntax **How to Establish Data and Index Buffers**

SET BUFND and BUFNI

```
{MVS|CMS} VSAM SET BUFND {n1|8}
{MVS|CMS} VSAM SET BUFNI {n2|1}
```

where:

n1

Is the number of data buffers. The default is BUFND=8.

n2

Is the number of index buffers. The default is BUFNI=1.

To determine how many buffers are in effect at any time, issue a VSAM SET query:

Syntax **Querying buffers currently in effect**

```
{MVS/CMS} VSAM SET ?
```

Alternate Indexes

FOCUS supports alternate indexes (keys) with VSAM key-sequenced data sources. Key-sequenced VSAM data sources consist of two elements: an index component and a data component. The data component contains the actual data records, while the index component is a key used to locate the data records in the file. Together, these two components comprise the “base cluster.”

Alternate indexes are separate, additional index structures that allow you to view the data records in a different sequence. For instance, you might usually access a personnel file by Social Security number, but occasionally need to retrieve records by job description. In this case, you could describe the job description field as an alternate index. Alternate indexes must be related to the base clusters they describe by a “path,” which is stored in a separate file.

Alternate indexes are VSAM structures that are created and maintained in the VSAM environment. You can describe them to FOCUS in the Master File, however, so that you can exploit them in the FOCUS environment.

These indexes offer improved efficiency. You can use one as an alternate, more efficient, retrieval path or you can take advantage of them indirectly, applying screening tests (IF...LT, IF...LE, IF...GT, IF...GE, IF...EQ, IF...FROM...TO, IF...IS) that translate into direct reads using the alternate index. You can also join data sources through an alternate index with the JOIN command.

It is not necessary to explicitly identify the indexed view in order to exploit the alternate index. FOCUS automatically selects the alternate index when one is described to FOCUS in the Master File.

Describing VSAM Data Sources to FOCUS

To use an alternate index in a TABLE request, simply provide an IF or WHERE test on the alternate index field that meets the above criteria. For example:

```
TABLE FILE CUST
PRINT SSN
WHERE LNAME EQ 'SMITH'
END
```

Note that if we define the LNAME field as an alternate index field, records in this file will be retrieved by last name and certain IF tests on the field LNAME will result in direct reads. If the alternate index fieldname were omitted, the primary key (if there was one) would be used for sequential or direct reads, and the alternate indexes would simply be treated as regular fields.

Describing Alternate Indexes

Alternate indexes must be described in Master Files with FIELDTYPE=I. The ALIAS of the alternate index field must be the filename allocated to the corresponding path name. Alternate indexes can be described as GROUPs if they consist of portions with dissimilar formats.

Remember that the ALIAS=KEY must be used to describe the primary key.

Consider the following example:

```
FILENAME=CUST, SUFFIX=VSAM,$
SEGNAME=ROOT, SEGTYPE=S0,$
GROUP=G, ALIAS=KEY, A10, A10,$
FIELD=SSN, SSN, A10, A10,$
FIELD=FNAME, DD1, A10, A10, FIELDTYPE=I,$
FIELD=LNAME, DD2, A10, A10, FIELDTYPE=I,$
```

In this example, SSN is a primary key and FNAME and LNAME are alternate keys (indexes). DD1 and DD2 are file names (ddnames) allocated to corresponding paths. CUST must be allocated to the base cluster.

Only one record type can be referenced in the request when alternate indexes are used, but the number of OCCURS segments is unrestricted.

The file name or ddname specified in the alias must be allocated to the path corresponding to the index. Note that the path name is different from both the cluster name and the alternate index name.

If you are uncertain of the path names and alternate indexes associated with a given base cluster, use the IDCAMS utility. (See the IBM manual entitled *Using VSAM Commands and Macros* for details.)

The following example demonstrates how to find the alternate index and path names associated with a base cluster named CUST.DATA:

Syntax **How to Find Alternate Index Names for a Cluster**

First, find the alternate index names (AIX) associated with the given cluster.

```
IDCAMS input
LISTCAT CLUSTER ENTRIES(CUST.DATA) ALL

IDCAMS output (fragments)
  CLUSTER ----- CUST.DATA
  ASSOCIATIONS
    AIX ----- CUST.INDEX1
    AIX ----- CUST.INDEX2
```

This provides the names of the alternate indexes (AIX): CUST.INDEX1 and CUST.INDEX2.

Syntax **How to Find Path Names Associated With a Given AIX Name**

Next, find path names associated with the given AIX name.

```
IDCAMS input:
  LISTCAT AIX ENTRIES (CUST.INDEX1 CUST.INDEX2) ALL

IDCAMS output (fragments):
  AIX -----CUST.INDEX1
  ASSOCIATIONS
    CLUSTER -- CUST.DATA
    PATH -----CUST.PATH1
  AIX -----CUST.INDEX2
  ASSOCIATIONS
    CLUSTER -- CUST.DATA
    PATH -----CUST.PATH2
```

This provides the path names: CUST.PATH1 and CUST.PATH2.

Advanced Topics

This section discusses two specialized techniques associated with describing external data sources.

Combining RECTYPE and OCCURS Segments

It is possible to describe external data sources that have two different types of descendant segments to FOCUS. The first segments are made up of multiple-occurrence fields within a single record of a given type. These you describe with the OCCURS attribute. You could also have descendant segments, describing separate, but related, records distinguished by a field that determines their record type. You describe this type of descendant segment using a RECTYPE field declaration in the Master File to describe the external field.

Describing VSAM Data Sources to FOCUS

The external data structure below has both types of segments. The first record, of type 01, contains several different sequences of repeating fields, all of which are described in the FOCUS structure as descendant segments, possessing an OCCURS attribute. The file also contains two separate types of records, (02 and 03) that contain information related to that in record type 01. This relationship was established in the external data structure. Here, we are simply concerned with describing that relationship to FOCUS.

The relationship between the records of various types is also expressed as a parent-child relationship. The children containing record types 02 and 03 will not have OCCURS attributes. They are distinguished from their parent by the field declaration: FIELDNAME=RECTYPE.

01	T1	N1	B1	B2	C1	C1	C1	D1	B1	B2	C1	C1	D1	D1						
D1	D1	D1	D1	D1																

02	E1
----	----

03	F1
----	----

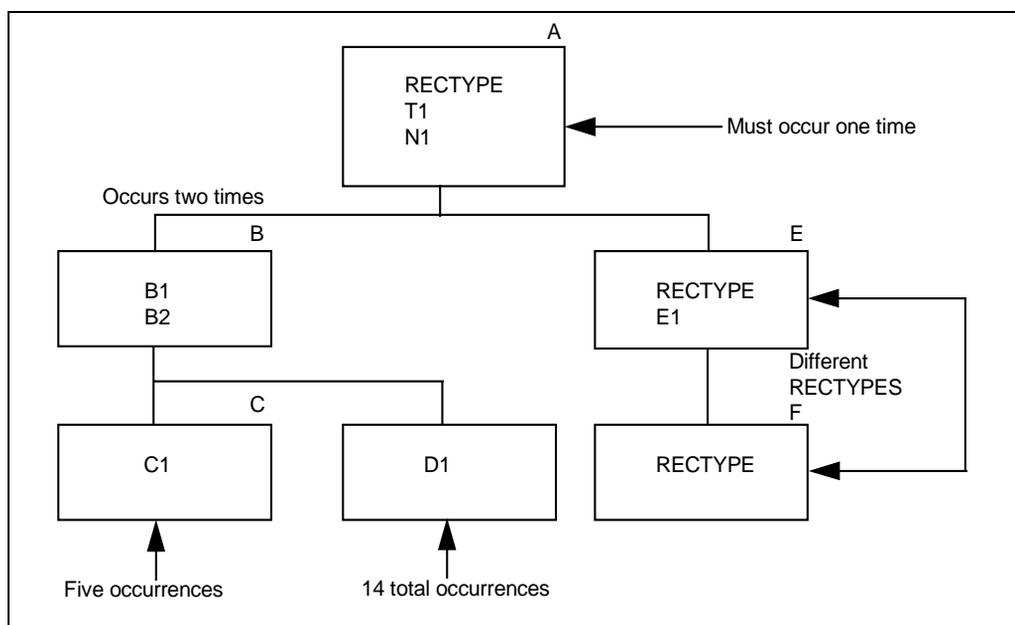
The Master File for this data source is:

```

FILENAME=EXAMPLE1, SUFFIX=FIX,$
SEGNAME=A, SEGTYPE=S0,$
FIELDNAME=RECTYPE, ALIAS=01, ACTUAL=A2, USAGE=A2,$
FIELDNAME=T1, ALIAS=, ACTUAL=A1, USAGE=A2,$
FIELDNAME=N1, ALIAS=, ACTUAL=A1, USAGE=A1,$
SEGNAME=B, SEGTYPE=S0, PARENT=A, OCCURS=VARIABLE,$
FIELDNAME=B1, ALIAS=, ACTUAL=I2, USAGE=I2,$
FIELDNAME=B2, ALIAS=, ACTUAL=I2, USAGE=I2,$
SEGNAME=C, SEGTYPE=S0, PARENT=B, OCCURS=B1,$
FIELDNAME=C1, ALIAS=, ACTUAL=A1, USAGE=A1,$
SEGNAME=D, SEGTYPE=S0, PARENT=B, OCCURS=7,$
FIELDNAME=D1, ALIAS=, ACTUAL=A1, USAGE=A1,$
SEGNAME=E, SEGTYPE=S0, PARENT=A,$
FIELDNAME=RECTYPE, ALIAS=02, ACTUAL=A2, USAGE=A2,$
FIELDNAME=E1, ALIAS=, ACTUAL=A1, USAGE=A1,$
SEGNAME=F, SEGTYPE=S0, PARENT=E,$
FIELDNAME=RECTYPE, ALIAS=03, ACTUAL=A2, USAGE=A2,$
FIELDNAME=F1, ALIAS=, ACTUAL=A1, USAGE=A1,$

```

It produces the following FOCUS data structure:



Segments A, B, C, and D all belong to the same physical record. Segments E and F are each stored on separate physical records in the external file.

Notes regarding this FOCUS data structure:

- Segments A, E, and F are different physical records, related through their record types. The record type attribute consists of certain prescribed values and is stored in a fixed location in the external records. FOCUS expects to read the records in a given order. If the first record does not have a RECTYPE of 01, the file is considered to be in error. The next record can have a RECTYPE of either 01 (in which case the first record is considered to have no descendants except the OCCURS descendants) or 02. A record with a RECTYPE of 03 can only follow a record with a RECTYPE of 02 (its parent).
- The OCCURS descendants all belong to the record whose RECTYPE is 01. (This is not a necessary condition; records of any type can have OCCURS descendants.) Note that the OCCURS=VARIABLE segment, Segment B, is the right-most segment within its own record type. If you look at the external data structure shown above, the pattern that makes up Segment B and its descendants (the repetition of fields B1, B2, C1, and D1) extends from the first mention of fields B1 and B2 to the end of the record.

Describing VSAM Data Sources to FOCUS

- Although fields C1 and D1 appear in separate segments, they are actually part of the repeating pattern that makes up the OCCURS=VARIABLE segment. Since they occur multiple times within Segment B, they are each assigned to their own descendant segment. The number of times field C1 occurs depends on the value of field B2. In the example, the first value of field B2 is 3, the second, 2. Field D1 occurs a fixed number of times, 7.

Reading Complex Data Sources With User-Written Procedures

There are various ways to read complex external data sources with user-written procedures. You can invoke a user exit and combine user-written code with data driver logical functions. These techniques are described Appendix B, *User Exits for Non-FOCUS Data Sources*.

You could also write your own routine to provide records to the FOCUS report writer from any source, which FOCUS will treat exactly as if they resided in a FOCUS data source. These user-written routines must be coded as subroutines in C, FORTRAN, COBOL or BAL, and they pass data to the FOCUS calling program through arguments in the subroutine. Appendix B provides an example of implementing this approach in FORTRAN. When taking this approach, be sure to refer to the PTF documentation in the READMEF file for OS/390 Installation concerning any applicable language environment (LE) fixes

Appendix B also demonstrates the use of a decompression exit (ZCOMP1) for working with compressed VSAM data sources and flat files.

3 Maintaining VSAM Data Sources

Topics:

- The IDCAMS Utility Program
- Accessing VSAM Data Sources
- Simultaneous Usage (SU) Mode for VSAM Data Sources in MVS
- Modifying Records: The MATCH and NEXT Statements
- Accessing Fields in Joined Data Structures With LOOKUP
- Modifying Multiple Data Sources: The COMBINE Facility
- Controlling the Current Record Position With REPOSITION
- Looping Through Files With REPEAT
- Modifying Variable-Length OCCURS Segments
- Modifying Fixed-Length OCCURS Segments
- Modifying Data Sources With Unrelated Records

Use FOCUS with the VSAM Write Data Adapter to create new VSAM data sources as well as updating existing ones. When creating a new one you must first calculate and allocate disk space for it following standard VSAM procedures before you initially load it. Use the DEFINE function of the IBM IDCAMS utility program to create it, as the FOCUS CREATE command, which is used to create FOCUS data sources and erase data in them, does not work with VSAM.

All FOCUS MODIFY and MAINTAIN features can be used with the VSAM Write Data Adapter. This chapter identifies particular aspects of the MATCH, NEXT, LOOKUP, COMBINE, FIND, REPOSITION, and REPEAT functions unique to VSAM.

MATCH and NEXT processing are also discussed herein. MATCH processing can only be performed on full key fields, on parts of the key fields, or on non-key fields. Unlike standard FOCUS processing of SO segment types, however, VSAM does not allow inclusion of new records with duplicate keys within key-sequenced data sets.

The IDCAMS Utility Program

All VSAM data sets, which are known as clusters, are defined and maintained using IDCAMS, which provides functions to create, copy, list, delete, and verify VSAM files. For detailed information concerning VSAM and IDCAMS functions, please refer to IBM's *VSAM Primer and Reference Manual*.

You must use IDCAMS to define the VSAM catalog before processing a VSAM data source. The catalog contains necessary information about the cluster, including its location and physical characteristics.

A database administrator or other authorized individual must allocate the VSAM catalog and grant users the authority to access (and/or update) the VSAM catalog or data sets. With write authority at the data set level, you can issue any IDCAMS command from VM/CMS and OS/390 or MVS.

The IDCAMS DEFINE function is used to describe VSAM attributes and record them in the master catalog. The volume serial number where each cluster resides is also recorded in the catalog at this time. All new VSAM data sets must be defined before you can load them with a MODIFY procedure.

If you declare an alternate index in your FOCUS Master File, you must also use a DEFINE to describe that alternate index and its path.

There are many optional DEFINE parameters. When defining VSAM files to FOCUS, we recommend using the IMBED and NOREPLICATE parameters to reduce access time, and SHAREOPTIONS (2) to ensure data integrity. The SHAREOPTIONS (2) parameter allows only one person to update a file at a time, but many can read it.

Accessing VSAM Data Sources

To access a VSAM file from an interactive OS/390 or MVS environment, it must be allocated to a ddname matching that of its Master File. In the OS/390 or MVS batch environment, you include the Master File ddname in your execution JCL.

In the VM/ESA environment, you issue a DLBL for the user catalog and the VSAM file.

In cases where alternate indexes are used, you must allocate or DLBL the path, or path dataset, for the alternate index cluster. The path maps the alternate index to its base cluster.

Simultaneous Usage (SU) Mode for VSAM Data Sources in MVS

Principles governing SU mode operations with FOCUS data structures also apply to SU mode with VSAM (VSAM/SU). In VSAM/SU mode, the VSAM data source is allocated to a job called Simultaneous Usage (SU). TSO userids and batch jobs running FOCUS are called source machines. Users on source machines send requests and transactions to the SU address space. VSAM files allocated to SU can only be updated by clients connected to it. Outside of VSAM/SU mode, multiple users can report against VSAM data sources in FOCUS, as well as in other languages, by simply allocating them with DISP=SHR.

No special requirements apply in defining VSAM clusters for SU. Standard efficiency considerations apply to these data sets: SHAREOPTION (1) or (2) is acceptable. Since SU allows only one transaction at a time against the VSAM data source, data integrity is always preserved.

Starting VSAM/SU Source Machines

TSO userids and batch jobs that access VSAM data sources on a FOCUS Database Server can be started with standard FOCUS CLISTs or JCL, with the following changes:

1. VSAM data sets and paths must not be allocated.
2. The FOCBMP data set allocated on the FOCUS Database Server must also be allocated on the source machine as well. It can be allocated to any ddname providing flexibility to access more than one FOCUS Database Server at a time.
3. During the FOCUS session, you must issue a USE command listing the ddnames of each VSAM cluster and path data set to be used during the session. For example:

```
USE
VSAM1 ON FOCBMP1
VSAM2 ON FOCBMP1
PATH2 ON FOCBMP1
VSAM9 ON FOCBMP2
END
```

The ddnames of VSAM clusters and path data sets must correspond to those used by SU. Ddnames of the FOCBMP data sets must correspond with those allocated by the client.

The USE command may be reissued with different parameters at any time during the FOCUS session.

For applications that only retrieve data, it is preferable not to go through the FOCUS Database Server, so do not issue USE commands for these sources, but allocate them as you would for a TSO user ID in single-user mode.

Querying the VSAM/SU FOCUS Database Server

To query FOCUS Database Server status, execute the XMISAMT FOCEXEC, selecting the INFO option. XMISAMT is on the FOCUS installation tape in the PDS FICCTL.DATA. Copy it to a central FOCEXEC library during installation.

Modifying Records: The MATCH and NEXT Statements

MATCH statements in MODIFY procedures select the first record with field values matching those that it provides. NEXT statements select the next record after the current position and make that the new current position.

MATCH or NEXT operations can select specific segments in a record, or specific segment instances in a multi-segment file structures. In this discussion, the logic that applies to record selection also applies to segment selection.

Current position depends upon the execution of MATCH and NEXT statements:

- At the start of a request, the current position is before the first record in the root segment.
- If a MATCH or NEXT statement selects a record, that becomes the current position.

When the full key is used in a MATCH statement, subsequent NEXT statements retrieve subsequent records, continuing to the end of the file.

When a partial key is used, the starting point is the partial key value. This type of operation results in selection of the subset of a file's records that contain the specified key value. The search continues until the partial-key value changes, resulting in a NONEXT condition.

Issuing a MATCH on a partial key selects only the first instance of the matched value. You then use NEXT to select subsequent matches.

While NEXT statements can update segments similarly to MATCH statements and follow the same rules, they are usually employed in read-only operations for displaying field values.

You can use NEXT statements in requests without Case Logic to modify or display the data in:

- The entire root segment.
- The first descendent segment.

Modifying Records: The MATCH and NEXT Statements

To modify or display data in all descendent segments, use Case Logic as described in your FOCUS documentation. Use the NEXT statement to update and display data in the root segment. This request displays all the PUBKEYs of the LIBRARY6 file we described in Chapter 2, *Describing VSAM Data Sources to FOCUS*:

```
MODIFY FILE LIBRARY6
.
.
.
NEXT PUBNO
ON NEXT TYPE "PUBLISHER NUMBER: <D.PUBNO"
ON NONEXT GOTO EXIT
.
.
.
DATA
```

Without resorting to Case Logic, a NEXT statement can only modify or display data in the first instance of a descendent segment.

This request uses Case Logic to display data from the LIBRARY6 file. When the descendent segments for a parent segment are exhausted, a NONEXT condition occurs.

```
MODIFY FILE LIBRARY6
PROMPT PUBNO
REPOSITION PUBNO
MATCH PUBNO
ON NOMATCH REJECT
ON MATCH TYPE "YOU ENTERED PUBLISHER #: <PUBNO"
ON MATCH GOTO CASENEXT

CASE CASENEXT
NEXT BOINKEY
ON NEXT TYPE
"THE PUBLISHER AND BINDING TYPE"
"IS <D.BOINKEY"
ON NEXT GOTO CASENEXT
ON NONEXT GOTO EXIT
ENDCASE

DATA
```

The MATCH statement selects a segment with a particular publisher number. The NEXT statement selects the segment with the group key information, that is, the publisher number along with the binding type indicator. The BOOK INFO segment is a descendent of the ROOT segment.

Accessing Fields in Joined Data Structures With LOOKUP

Used the MODIFY LOOKUP function to retrieve data values from cross-referenced data structures. To use LOOKUP with VSAM data sources, issue a JOIN command to link them and then issue a LOOKUP on the key field or alternate index.

LOOKUP is required because, unlike TABLE requests, MODIFY requests cannot freely read cross-referenced files. The LOOKUP function enables you to issue requests containing retrieved data values in computations and messages.

LOOKUP is valid in both COMPUTE and VALIDATE statements. Requests can use the data retrieved in the computations or validations within MODIFY procedures. Use LOOKUP to locate values in any data sources that can be linked with the JOIN command.

You cannot modify data in the cross-referenced file. To modify files other than the host file in a request, use the COMBINE command described in the next section.

Syntax

Using the LOOKUP Function

```
rcode = LOOKUP(field);
```

where:

`rcode`

Is the variable (you specify) to receive a return code. The return code value is 1 if LOOKUP can locate a cross-referenced segment instance or 0 if it cannot.

`field`

Is the name of the field to be retrieved in the cross-referenced file. LOOKUP retrieves a single field at a time. Each requires a separate LOOKUP.

Syntax note: no space is permitted between LOOKUP and the left parentheses. LOOKUP can exist by itself or as part of a larger expression. If it is used by itself, it must end in a semicolon.

The extended syntax of the LOOKUP function, described in your FOCUS documentation, does not apply to the VSAM Interface. You can only retrieve exact match values.

When using LOOKUP, the MODIFY request reads a transaction for the host field value. LOOKUP then searches the cross-referenced segment for an instance with this value in the cross-referenced field:

- If no such instances exist, the rcode variable is set to 0. If you use the field specified by the LOOKUP function in the request, that field assumes a value of blank if alphanumeric or 0 if numeric.
- If a record containing the transaction value is found, the function sets the rcode variable to 1 and retrieves the LOOKUP field value from the first matching segment found. All field values in the segment are then available for COMPUTE or VALIDATE operations.

Modifying Multiple Data Sources: The COMBINE Facility

To update multiple unrelated data sources with a single transaction, use the COMBINE command, which enables you to modify two or more VSAM data sources in one MODIFY request. The command combines the logical structures of the two sources into one structure while leaving the physical structures untouched.

COMBINE structures, which can contain up to 16 files and 64 segments, last for the duration of a FOCUS session or until you enter another COMBINE command. Only one combined structure can exist at a time.

Syntax

COMBINE Command

```
COMBINE FILES  file1 [PREFIX string1] [AND]
                file2 [PREFIX string2] [AND] ...
                filen [PREFIX stringn] AS newname
```

where:

`file1 ... filen`

Are the names of the VSAM files to be modified (up to 16 files).

`PREFIX stringn`

Adds the specified string as a prefix for all fieldnames in the file, thereby avoiding the possibility of duplicate fieldnames in combined files.

`AND`

Is an optional word included to enhance readability.

`newname`

Is the name of the combined structure to be used in the subsequent MODIFY and CHECK commands.

Note: You can type the command on as many lines as necessary.

In your MODIFY procedure, use newname as follows:

```
MODIFY FILE newname
```

Placing statements pertaining to each file in different cases helps clarify the request logic.

Accessing Fields in Combined Data Sources With the FIND Function

The VSAM Write Data Adapter supports the MODIFY FIND function for verifying the existence of incoming data values in VSAM data sources. The fields named must be keys or alternate indexes.

FIND sets a temporary field to 1 if a value exists or 0 if one does not. You can test this value and branch on it using Case Logic.

To use FIND, include the VSAM data source in a COMBINE statement. The MODIFY procedure acts on the combined structure. Any fieldname used with the FIND function must be unique across the combined sources.

Syntax **FIND Function**

```
rfield = FIND(fieldname AS dbfield IN file);
```

where:

`rfield`

Is a variable you specify to receive a return code value. This value is 1 if the FIND function can locate the data value or 0 if it cannot.

`fieldname`

Is the full name (not the alias or a truncation) of the incoming tested field.

`dbfield`

Is the full name (not the alias or a truncation) of the VSAM field containing the values being compared with the incoming data field. This must be a primary key or an alternate index.

`file`

Is the name of the VSAM data source where the dbfield resides.

Note: There is no space between FIND and the left parenthesis.

Controlling the Current Record Position With REPOSITION

With each successful MATCH in MODIFY process, the record containing the matched value becomes your current position within the file.

In NOMATCH situations, the current position will be the end of the file. Because FOCUS does not recognize VSAM record keys, the current position does not return to the beginning of the file after each pass. This creates a possibility that processing could terminate unexpectedly, or, that a NOMATCH condition might occur despite the existence of a matching key value in the file. The solution is to include the REPOSITION statement in all MODIFY procedures to change your current position and prevent abnormal terminations.

Syntax **Establishing Current Position Using REPOSITION Statements**

```
REPOSITION field
```

where:

`field`

Is any field of any segment.

REPOSITION enables you to change your current position in the data source, either returning to the beginning, or to the beginning of a record, or to the beginning of a segment chain within a record (that is, the parent segment, the parent's parent, and so on to the root segment). From there, you can search the segment chain from the beginning.

In the following example, REPOSITION is used to return the current position of the STUDENT field to the beginning of the file, enabling you to reenter data for STUDENT and LAST NAME.

```
MODIFY FILE VSAM2
REPOSITION STUDENT
MATCH STUDENT
ON NOMATCH INCLUDE
ON NOMATCH GO TO NEWREC
ON MATCH GO TO OLDREC
.
.
.
```

After executing REPOSITION, the current position depends on your MATCH or NEXT logic. When a NOMATCH or NONEXT condition exists, you tell FOCUS how to proceed by specifying a fieldname to tell it how to process the current data source.

Looping Through Files With REPEAT

MODIFY requests using FIDEL usually prompt for a key field value and then use that to retrieve a segment instance. After modifying that instance, you enter a new key field value to retrieve the next instance. In this way, you can modify one segment instance at a time.

The REPEAT statement enables you to process records with segments that have multiple-occurrences. With REPEAT, you can add, update, or delete several segment instances on one screen.

Multi-record processing causes the request to retrieve multiple segment instances before FIDEL displays instance values. Each time the request retrieves a segment instance, it stores its values in a work area in memory called the Scratch Pad Area. The request continues retrieving instances until the specified number is reached.

After retrieving the instances, FIDEL reads the instance values from the Scratch Pad Area and displays them all on one screen. You can then update them and transmit the updated values back to the VSAM data source.

You can also design requests to add several segments at a time or to both update existing segments and add new ones on the same screen.

This section describes multiple-occurrence record processing using REPEAT. One REPEAT statement collects record instances and loads them into the Scratch Pad Area (SPA); another REPEAT statement retrieves records from the SPA and uses them to modify the database. This method does not require knowledge of how the records are stored in the SPA. You must process them sequentially, however, and you can process only one at a time.

Refer to your FOCUS documentation for additional information about the REPEAT statement.

Syntax **Looping With the REPEAT statement**

```
REPEAT {*/count} [TIMES] [MAX limit] [NOHOLD]  
phrases
```

```
                  :  
ENDREPEAT
```

where:

count

Is an integer or temporary integer field that specifies the number of times the request executes REPEAT. The value (from 0 to 32,767) should not be smaller than the number of segment instances to be displayed on the FIDEL screen.

If count is 0, the REPEAT does not execute (this allows you to skip a REPEAT if you use a temporary field for this parameter).

If count is an asterisk, the REPEAT executes indefinitely. To end it, code a GOTO ENDREPEAT.

Once REPEAT begins execution, the count value cannot be changed.

TIMES

Is an optional word used to improve readability.

limit

Is an integer specifying the maximum number of times to execute the REPEAT. Specify this only when using a temporary field for the count parameter.

NOHOLD

Is an option that allows you to use REPEAT as a simple loop that executes a group of MODIFY statements repeatedly.

phrases

Are MODIFY statements to be executed within the REPEAT statement. Each must begin on a new line.

ENDREPEAT

Ends the REPEAT statement and must appear on a line by itself.

REPEAT statements can stand by themselves or can be part of ON MATCH, ON NOMATCH, ON NEXT, or ON NONEXT phrases in MATCH or NEXT statements. The following example loads multiply occurring segments of the VSAM1 data source.

```
CASE ADDCASE
REPEAT 4 TIMES
COMPUTE QTR_NUM=REPEATCOUNT;
MATCH QTR_NUM
ON MATCH UPDATE GRADE QTR_AVE
ON MATCH GO TO ENDREPEAT
ENDREPEAT
ENDCASE
```

REPEAT statements cannot be nested - one must end before the next begins.

Also, note use of the phrase GOTO ENDREPEAT for branching to the end of the REPEAT statement. REPEATCOUNT is a FOCUS variable that is incremented each time the REPEAT is executed.

Modifying Variable-Length OCCURS Segments

The VSAM Write Data Adapter supports both types of variable length records (e.g., OCCURS=fieldname and OCCURS=VARIABLE). You can issue INCLUDE, UPDATE, and DELETE statements in MODIFY procedures against these segments using standard MATCH and NEXT logic.

If you specify OCCURS=fieldname, FOCUS automatically updates that field each time occurrences are added or deleted. Do not include logic in your procedure for maintaining this field, or include a counter field as part of the transaction data.

Response may be better using the OCCURS=VARIABLE segment attribute because VSAM controls most of the processing. However, only one segment per data source can be defined this way and that must be the last segment in the Master File to ensure that FOCUS interprets the data correctly.

For some applications, OCCURS=fieldname may be more beneficial, because the counter is maintained by the MODIFY processing and you have flexibility for using more than one OCCURS segment.

Maintaining VSAM Data Sources

You could use the following procedure to INCLUDE, UPDATE, OR DELETE data in variable-length records. It refers to the file VSAM3.

```
MODIFY FILE VSAM3
COMPUTE
IDTEMP/A9=;
LNTEMP/A15=;
CRTFORM LINE 1

*****
" *                RIVERDALE JUNIOR HIGH                * "
" *                STUDENT GRADES MAINTENANCE            * "
" *                                                        * "
" *                PLEASE ENTER THE FOLLOWING INFORMATION: * "
" *                                                        * "
" *                STUDENT ID:<IDTEMP/09                 * "
" *                LAST NAME:<LNTEMP/15                 * "
" *                                                        * "
" *                ENTER=CONTINUE PFl=EXIT              * "
*****

COMPUTE
STUDENT=IDTEMP;
LAST_NAME=LNTEMP;
REPOSITION STUDENT
MATCH STUDENT
ON NOMATCH TYPE " NEW STUDENT: <STUDENT "
ON NOMATCH INCLUDE
ON NOMATCH GOTO NEWREC
ON MATCH TYPE " UPDATING STUDENT: <STUDENT "
ON MATCH GOTO OLDREC
```

The next section determines if the input is for a new or existing STUDENT. It displays a message and then branches to the appropriate case.

```
CASE NEWREC
COMPUTE
STUDENT=IDTEMP;
LAST_NAME=LNTEMP;
MATCH STUDENT
ON NOMATCH TYPE "IMPROPER CONDITION IN NEWREC"
ON NOMATCH REJECT
ON MATCH CRTFORM LINE 1 TYPE 3
```

The following CRTFORM displays students' last names and identification numbers. It then enables you to enter up to five occurrences of COURSE, GRADE, and QTR TAKEN data. The VSAM3 file is defined with a variable number of child segments.

```

*****
**                                RIVERDALE JUNIOR HIGH                                **
**                                                                                   **
**                                                                                   **
**                                LAST NAME:<D.ID FIRST NAME:D.FN <69                    **
**                                                                                   **
**    COURSE <25   GRADE      <50   QUARTER_TAKEN <69 **
**    ----- <25   -----    <50   ----- <69 **
** <CN(001) <25 <GRADE(001) <50 <QTR_TAKEN(001) <69 **
** <CN(002) <25 <GRADE(002) <50 <QTR_TAKEN(002) <69 **
** <CN(003) <25 <GRADE(003) <50 <QTR_TAKEN(003) <69 **
** <CN(004) <25 <GRADE(004) <50 <QTR_TAKEN(004) <69 **
** <CN(005) <25 <GRADE(005) <50 <QTR_TAKEN(005) <69 **
**                                                                                   **
**                                                                                   **
**    ENTER=UPDATE          PF1=EXIT          PF2=HOME          **
*****

```

A MATCH is done on the COURSE field. If there is no COURSE information, the occurrence is loaded with the new data, updating the existing course data.

```

ON MATCH UPDATE FN
ON MATCH REPEAT 5 TIMES
GETHOLD
ACTIVATE COURSE GRADE QTR_TAKEN
IF COURSE EQ ' ' THEN GOTO EXITREPEAT;
MATCH COURSE
ON MATCH UPDATE GRADE QTR_TAKEN
ON MATCH GOTO ENDREPEAT
ON NOMATCH INCLUDE
ON NOMATCH GOTO ENDREPEAT
ENDREPEAT
GOTO TOP
ENDCASE

```

Maintaining VSAM Data Sources

If the student is in the file, processing continues with the OLDREC case, where we hold existing COURSE data in the Scratch Pad Area. The procedure can be repeated up to five times because a variable number of segment occurrences are anticipated.

```
CASE OLDREC
COMPUTE
DELT/Al=' ';
MATCH STUDENT LAST NAME
ON NOMATCH TYPE "IMPROPER CONDITION IN OLDREC"
ON NOMATCH REJECT
ON MATCH REPEAT 5 TIMES
NEXT COURSE
ON NEXT HOLD COURSE GRADE QTR_TAKEN DELT
ON NEXT GOTO ENDREPEAT
ON NONEXT GOTO EXITREPEAT
ENDREPEAT
```

To delete a course, type a D in the field DELT on the CRTFORM.

```
*****"
"*          RIVERDALE JUNIOR HIGH          *"
"*                                          *"
"*                                          *"
"*          LAST NAME:<D.ID FIRST NAME:D.FN <69  *"
"*                                          *"
"*  COURSE <25  GRADE      <50  QUARTER_TAKEN <66D <69  *"
"*  ----- <25  -----   <50  ----- <66- <69  *"
"* <T.CN(001) <25 <T.GRADE(001) <50 <T.QTR_TAKEN(001)<66 <T.DELT(001*"
"* <T.CN(002) <25 <T.GRADE(002) <50 <T.QTR_TAKEN(002)<66 <T.DELT(002*"
"* <T.CN(003) <25 <T.GRADE(003) <50 <T.QTR_TAKEN(003)<66 <T.DELT(003*"
"* <T.CN(004) <25 <T.GRADE(004) <50 <T.QTR_TAKEN(004)<66 <T.DELT(004*"
"* <T.CN(005) <25 <T.GRADE(005) <50 <T.QTR_TAKEN(005)<66 <T.DELT(005*"
"*                                          *"
"*  ENTER=UPDATE      PF1=EXIT      PF2=HOME      *"
*****"
```

This section updates the segment if data is marked for deletion and then branches to case DELCASE.

```
ON MATCH UPDATE FN
ON MATCH REPEAT 5 TIMES
GETHOLD
ACTIVATE COURSE GRADE QTR_TAKEN
IF COURSE EQ ' ' THEN GOTO EXITREPEAT;
MATCH COURSE
ON MATCH UPDATE GRADE QTR_TAKEN
ON MATCH IF DELT EQ 'D' THEN PERFORM DELCASE;
ON MATCH GOTO ENDREPEAT
ON NOMATCH INCLUDE
ON NOMATCH GOTO ENDREPEAT
ENDREPEAT
GOTO TOP
ENDCASE
```

This section deletes a child segment.

```

CASE DELCASE
ACTIVATE COURSE GRADE QTR_TAKEN
MATCH COURSE
ON MATCH DELETE
ON MATCH TYPE " COURSE: <COURSE DELETED "
ON NOMATCH REJECT
ENDCASE

DATA VIA FIDEL
END

```

Modifying Fixed-Length OCCURS Segments

Data sources with OCCURS=n segments are maintained using standard MATCH and NEXT logic in a MODIFY procedure.

The following Master File illustrates a VSAM data source with an OCCURS=n segment:

```

FILENAME=VSAMFILE , SUFFIX=VSAM
SEGNAME=ROOT , SEGTYPE=SO .S
GROUP=KEY INFO , ALIAS=KEY , USAGE=A13 , ACTUAL=A15 , $
FIELD=STUDENT , ALIAS=LN , USAGE=A9 , ACTUAL=A9 , $
FIELD=BIRTHDAY , ALIAS=BD , USAGE=I6YMD , ACTUAL=A6 , $
FIELD=FIRST NAME , ALIAS=FN , USAGE=A15 , ACTUAL=A15 , $
SEGNAME=REPSEG , PARENT=ROOT , OCCURS=9 , SEGTYPE=SO , $
FIELD=COURSE , ALIAS=CN , USAGE=A9 , ACTUAL=A9 , $
FIELD=GRADE , ALIAS=GD , USAGE=A1 , ACTUAL=A1 , $
FIELD=QTR_TAKEN , ALIAS=QA , USAGE=A2 , ACTUAL=A2 , $

```

A MODIFY procedure for loading a new student record automatically fills in the OCCURS segment with zeros and spaces as appropriate. You cannot use INCLUDE or DELETE to maintain fixed-length, multiply-occurring segments. Instead, use a MODIFY procedure with an UPDATE statement to load them. If you specify the ORDER field in your Master File, you can use that with MATCH or NEXT commands to retrieve appropriate occurrence of the repeating fields.

This procedure illustrates how to maintain the VSAMFILE defined above. The first section determines whether the input data is for a new or existing record.

```

MODIFY FILE VSAMFILE
FIXFORM ON BIGFIX STUDENT/9 BIRTHDAY/6
REPOSITION STUDENT
MATCH STUDENT
ON NOMATCH INCLUDE
ON MATCH/NOMATCH GOTO ANYREC

```

Maintaining VSAM Data Sources

When including a new root segment, the OCCURS=n segments are all filled with zeros and spaces as appropriate. Therefore, all actions against repeating fields are considered updates and can be processed by the ANYREC case.

```
CASE ANYREC
COMPUTE
DELT/A1 = ' ';
MATCH STUDENT BIRTHDAY
ON NOMATCH REJECT
ON MATCH REPEAT 9 TIMES
NEXT COURSE
ON NEXT HOLD COURSE GRADE QUARTER_TAKEN DELT
ON NEXT GOTO ENDREPEAT
ON NONEXT GOTO EXITREPEAT
ENDREPEAT
```

The next CRTFORM displays the student's last name, first name, and COURSE information. You can update the student's name and add or update up to nine occurrences of COURSE information. You can also delete an occurrence by placing a D in the field DELT.

```
ON MATCH CRTFORM LINE 1
*****
" *                TECH REP HIGH SCHOOL                * "
" *                -----                * "
" *   LAST NAME:<D.STUDENT   FIRST NAME:<T.FN>          * "
" *                                                                * "
" *   COURSE           GRADE           QUARTER_TAKEN    * "
" *   -----           -----           -----        * "
" * <T.COURSE(001) <T.GRADE(001) <T.QTR_TAKEN(001) <T.DELT(001) * "
" * <T.COURSE(002) <T.GRADE(002) <T.QTR_TAKEN(002) <T.DELT(002) * "
" * <T.COURSE(003) <T.GRADE(003) <T.QTR_TAKEN(003) <T.DELT(003) * "
" * <T.COURSE(004) <T.GRADE(004) <T.QTR_TAKEN(004) <T.DELT(004) * "
" * <T.COURSE(005) <T.GRADE(005) <T.QTR_TAKEN(OOS) <T.DELT(005) * "
" * <T.COURSE(006) <T.GRADE(006) <T.QTR_TAKEN(006) <T.DELT(006) * "
" * <T.COURSE(007) <T.GRADE(007) <T.QTR_TAKEN(007) <T.DELT(007) * "
" * <T.COURSE(008) <T.GRADE(008) <T.QTR_TAKEN(008) <T.DELT(008) * "
" * <T.COURSE(009) <T.GRADE(009) <T.QTR_TAKEN(009) <T.DELT(009) * "
" *                                                                * "
" *                                                                * "
" *   ENTER=UPDATE           PF1=EXIT           PF2=HOME          * "
*****
```

This section can either update the existing record or include new values. If data is marked for deletion, processing branches to DELCASE.

```

ON MATCH UPDATE FN
ON MATCH REPEAT 9 TIMES
GETHOLD
IF DELT EQ 'D' THEN PERFORM DELCASE;
NEXT COURSE
ON NEXT UPDATE COURSE GRADE QTR_TAKEN
ON NEXT GOTO ENDREPEAT
ENDREPEAT
GOTO TOP
ENDCASE

```

Delete values by updating the COURSE, GRADE, and QTR_TAKEN fields with blanks.

```

CASE DELCASE
COMPUTE
COURSE = ' '; GRADE = ' '; QTR_TAKEN = ' ';
NEXT COURSE
ON NEXT UPDATE COURSE GRADE QTR_TAKEN
GETHOLD
ENDCASE

DATA
END

```

Modifying Data Sources With Unrelated Records

During MODIFY processing, FOCUS determines and maintains the appropriate record type based on the fieldnames in the transactions. Do not include RECTYPE as an input data fields because FOCUS automatically inserts the correct RECTYPE value when a new record is included.

Consider a sporting goods store that sells tents and sails and maintains its inventory information in one file. This data source has two types of records: one containing tent information and one containing sail information.

```
FILE=SPORTS ,SUFFIX=VSAM , $
SEGNAME=DUMMY , $
FIELD= , , USAGE=A1 ,ACTUAL=A1 , $
SEGNAME=SAILSEG ,SEGTYPE=SO ,PARENT=DUMMY , $
GROUP=SAILKEY ,ALIAS=KEY ,USAGE=A13 ,ACTUAL=A13 , $
FIELD=SAIL_NAME ,ALIAS=SNA ,USAGE=A3 ,ACTUAL=A3 , $
FIELD=SAIL NUMBER ,ALIAS=SNO ,USAGE=A10 ,ACTUAL=A10 , $
FIELD=RECTYPE ,ALIAS=S ,USAGE=A1 ,ACTUAL=A1 , $
FIELD=S TYPE ,ALIAS=ST ,USAGE=A6 ,ACTUAL=A6 , $
FIELD=S WIDTH ,ALIAS=SW ,USAGE=I1 ,ACTUAL=I2 , $
FIELD=S LENGTH ,ALIAS=SL ,USAGE=I2 ,ACTUAL=I2 , $
FIELD=S QUANTITY ,ALIAS=SQ ,USAGE=I8 ,ACTUAL=I4 , $
FIELD=S W PRICE ,ALIAS=SP ,USAGE=P8.2 ,ACTUAL=P5 , $
FIELD=S R PRICE ,ALIAS=SP ,USAGE=P8.2 ,ACTUAL=PS , $
SEGNAME=TENTSEG ,SEGTYPE=SO ,PARENT=DUMMY , $
GROUP=TENT KEY ,ALIAS=KEY ,USAGE=A13 ,ACTUAL=A13 , $
FIELD=MANUFACTURER ,ALIAS=MFG ,USAGE=A5 ,ACTUAL=A5 , $
FIELD=STYLE NO ,ALIAS=SNO ,USAGE=A6 ,ACTUAL=A6 , $
FIELD=COLOR ,ALIAS=CO ,USAGE=A2 ,ACTUAL=A2 , $
FIELD=RECTYPE ,ALIAS=T ,USAGE=A1 ,ACTUAL=A1 , $
FIELD=T HEIGHT ,ALIAS=TH ,USAGE=I2 ,ACTUAL=I2 , $
FIELD=SLEEP SZ ,ALIAS=SS ,USAGE=I2 ,ACTUAL=I2 , $
FIELD=T QUANTITY ,ALIAS=TQ ,USAGE=I8 ,ACTUAL=I4 , $
FIELD=T W PRICE ,ALIAS=TP ,USAGE=P8.2 ,ACTUAL=P5 , $
FIELD=T R PRICE ,ALIAS=TP ,USAGE=P8.2 ,ACTUAL=P5 , $
```

This MODIFY procedure updates the inventory data source when new shipments of tents and sails arrive. Tent records end at byte 35, sail records at byte 44. The CHECKFLD tests for blanks in bytes 36 through 38. When these three bytes are blank, processing branches to the TENTPROC case. If the three bytes contain data, processing branches to case SAILPROC. The transaction data source is allocated to ddname TENTSAIL

```
MODIFY FILE SPORTS
COMPUTE
CHECKFLD/A3=;
T_NEWQUAN/I6=;
S_NEWQUAN/I6=;
FIXFORM X35 CHECKFLD/3
IF CHECKFLD EQ ' ' THEN GOTO TENTPROC
ELSE GOTO SAILPROC;
```

Modifying Data Sources With Unrelated Records

```
CASE TENTPROC
FIXFORM X-38 MANUFACTURER/5 STYLE NO/6 COLOR/2 T_HEIGHT/2
FIXFORM SLEEP_SZ/2 T_W_PRICE/6 T_R_PRICE/6 T_NEWQUAN/I6 X9
MATCH MANUFACTURER STYLE_NO COLOR
ON MATCH COMPUTE
T_QUANTITY=D.T_QUANTITY + T_NEWQUAN;
ON MATCH UPDATE HEIGHT SLEEP_SZ T_W_PRICE T_R_PRICE T_QUANTITY
ON NOMATCH INCLUDE
ON MATCH/NOMATCH GOTO TOP
ENDCASE

CASE SAILPROC
FIXFORM X-38 SAIL_NAME/3 SAIL_NUMBER/10 S_TYPE/6 S_WIDTH/2
FIXFORM S_LENGTH/2 S_STYLE/3 S_W_PRICE/P6 S_R_PRICE/P6 S_NEWQUAN/I6
MATCH SAIL_NAME SAIL_NUMBER
ON MATCH COMPUTE
S_QUANTITY=D,S_QUANTITY + S_NEWQUAN;
ON MATCH UPDATE S_TYPE S_WIDTH S_LENGTH S_STYLE S_W_PRICE S_R_PRICE
ON MATCH UPDATE S_QUANTITY
ON NOMATCH INCLUDE
ON MATCH/NOMATCH GOTO TOP
ENDCASE

DATA ON TENTSAIL
END
```

A Debugging Techniques

Since several products control the processing in your environment, you could receive error messages generated by FOCUS, by VSAM, or by your operating system. Start by evaluating the number of any FOCUS error messages.

If the FOCUS error message number is under 1000, check your procedure for standard FOCUS syntax errors. If the number is over 1000, the error is either a data-adapter-related or user-related problem.

- If there is NO error message, review your FOCUS MODIFY procedure.
- If the error occurs at the beginning of program execution, check allocations and disk access.
- If the output data was truncated, first inspect the field lengths in the Master File and make sure that they match the record lengths in the VSAM data source.
- Verify that the Master File adheres to specified format rules.
- Check all GROUP definitions and confirm that ALIAS=KEY and that all USAGE and ACTUAL fields are in alpha format.
- If there is an alternate index, see that a path was allocated for it.
- If processing is extremely slow, use the IDCAMS LISTCAT function to determine whether there are too many CA or CI splits.

B User Exits for Non-FOCUS Data Sources

Topics:

- The Dynamic and Re-Entrant GNTINT Private User Exit
- User-coded Data Access Modules
- Re-Entrant VSAM Compression Exit: ZCOMP1

This appendix describes three ways to read non-FOCUS data sources with user-written procedures.

The Dynamic and Re-Entrant GNTINT Private User Exit

The GNTINT Interface, which is used for accessing VSAM and QSAM structures, has a user exit that can be invoked as an alternative to the lowest-level retrieval routines of the VSAM Write Data Adapter. In such cases, the Master File would specify SUFFIX=VSAM or SUFFIX=FIX. The user exit facilitates combining user-written code with GNTINT logical retrieval functions, such as record selection logic, treatment of missing records in multi-record files, JOINS between various types of files, and so forth, devoid of dependence on internal FOCUS structures. Major features of the exit are:

1. Through a CONTEXT parameter, GNTINT supports reentrancy, reducing storage requirements while enhancing performance.
2. Multiple concurrent exit processors are supported through an Access File which permits the naming of user exit modules on a per Master File basis.
3. The GNTINT user exit is called dynamically at execution, thus avoiding the need to modify FOCUS after each upgrade or maintenance run. There is no more need for link-edits to FOCUS.
4. An initialization call enables the exit code to perform initial housekeeping.
5. The QUALIF parameter supports processing options: (O) OPEN file, (R) OPEN request (position), (C) CLOSE, and (F) Fin of FOCUS, in addition to the (S), (G), and (E) read options .
6. The exit supports multiple positions in the same file.

Functional Requirements

Functionally, the private user exit code replaces retrieval calls typically used against, but not limited to, key-sequenced VSAM files, and can be used against any data source that can be represented as such. The user code need not deal with structures represented by OCCURS clauses, nor with translation of FOCUS IF conditions into lower-level criteria, functions now performed by GNTINT Interface logic.

The user-written code must be able to do the following:

1. Obtain a record, given a full value of the key. Keys are presumed to be unique (direct reads).
2. Obtain a record greater than or equal to a given value of the full or partial key (generic read).
3. Obtain the next logical record, starting from the current position in the file (sequential read). Successive sequential reads must return records in ascending key sequence (bit by bit).
4. Direct and generic reads that retrieve records must establish the starting position in the file for subsequent sequential reads. Direct reads that fail to retrieve the requested records need not establish these positions.
5. For the open file request (O), it must logically or physically open the file.
6. If a logical end-of-file condition results from a sequential read, an end-of-file signal must be returned. Subsequent sequential reads must return end-of-file signals rather than error indications, for example, when processing a JOIN.
7. A 'close' function is required that releases all resources and gives up the current position in the file, enabling subsequent open requests to succeed.
8. Successive close calls must be innocuous (be prepared to close files that are already closed).
9. A unique area must be obtained, for example, GETMAIN, and maintained using the CONTEXT parameter on a per DDNAME basis.
10. The code must be serially reusable and reentrant. It must be linked AMODE 31.

Implementation

The Dynamic GNTINT User Exit is linked as a separate module and loaded or called from FOCUS.

The Master File

The Master File for data to be accessed using this user exit is exactly the same as the description of any other data source read by the GNTINT Interface except that the SUFFIX must specify PRIVATE. All other READ ONLY features of the GNTINT Interface are fully supported.

Example

Sample Master File

```
FILE=filename, SUFFIX=PRIVATE,$
SEGNAME=ROOT , SEGTYPE=S0,$
GROUP=keyname , ALIAS=KEY , USAGE=xx, ACTUAL=xx , $
FIELD=fieldname1, ALIAS=aliasname1, USAGE=xx, ACTUAL=xx , $
FIELD=fieldname2, ALIAS=aliasname2, USAGE=xx, ACTUAL=xx , $
```

Note: SUFFIX=PRIVATE. No other options will invoke the exit.

The Access File

An Access File is required and provides a pointer to the actual name of the private exit. The PDS used for the Access File must be allocated to DDNAME ACCESS.

Example

Sample Access File

```
MODNAME=pgmname , $
```

Example

Allocating an Access File

```
//ACCESS DD DSN=access.file.pds.name,DISP=SHR
```

Calling Sequence

The user-coded retrieval routine is written as a standalone program. There are no limitations to program name other than standard IBM rules. We recommend writing the program in a language that supports reentrancy, such as ASSEMBLER or C. The parameter list is as follows:

NCHAR

Full-word binary integer, posted by the exit. A positive number indicates the length in bytes of the record obtained; zero indicates no record or end-of-file (there is no difference). Must be non-zero for every successful read. Used by read options (S), (E), and (G). Do not modify the pointer. Instead, modify the location addressed by the pointer.

User Exits for Non-FOCUS Data Sources

DDNAME

8-byte character argument, posted by GNTINT, corresponding to the FOCUS file name for the generated report. For example, TABLE FILE CAR results in DDNAME CAR, left-justified and blank-padded. Used for all options except (F). Do not modify this parameter.

ABUFF

Full-word binary integer, posted by the exit; the absolute address of the record obtained by this call. Used with all read options (S), (E), and (G). Do not modify this pointer. Instead, modify the location addressed by the pointer.

RC

Full-word binary integer return code, posted by the exit. Zero indicates no error; non-zero indicates some type of error. Used with all options. Do not modify the pointer. Instead, modify the location addressed by the pointer.

KEY

Full-word binary integer posted by GNTINT, containing the full value of the key for direct or generic reads. Not significant for sequential reads. The key value is left justified and less than 255 bytes long.

Note that the exit must know the true key length and its format. Used with options (E) and (G). Do not modify this parameter.

OPT

Four-byte character argument, posted by GNTINT, indicating the type of call. Do not modify this parameter. The OPTions are as follows:

READ OPTIONS

- 'S ' =sequential read.
- 'E ' =direct read (EQ).
- 'G ' =generic read (GE).

CONTROL OPTIONS

- 'O ' =OPEN file.
 - 'R ' =OPEN request (position) used for recursive JOINS.
 - 'C ' =CLOSE file.
 - 'F ' =FIN for FOCUS -- final housekeeping should be done here.
- These control and read arguments must include three trailing blanks.

CONTEXT

Full-word binary integer, posted by GNTINT, which points to a work area described below. Do not modify this parameter.

Note: The parameters passed to the exit are not delimited with the final parameter having the high-order bit set on. Make sure that your program does not scan for this high-order bit.

Work Area Control Block

EYECATCH

An 8-character string containing the literal 'PRIVATE '.

PFMCB

Full-word binary integer, posted by the exit. Generally set during OPTion (O) processing by the exit program and returned unchanged by subsequent GNTINT calls. This parameter is generally used to point to the dynamic work areas used to maintain reentrancy.

PFACB

Full-word binary integer, posted by the exit. Generally set during OPTion (O) processing by the exit program and returned unchanged by subsequent GNTINT calls for OPTions (C), (R), (E), (G), and (S) and is unique by DDNAME. This is generally used as a physical file context. This parameter is not valid for OPTion (F).

PFRL

Full-word binary integer, posted by the exit. Generally set during option (OPT=R) processing by the exit program and returned unchanged by subsequent GNTINT calls for OPTions (E), (G), and (S) and is unique for each view within the above PFACB parameter. This is generally used for logical file context.

This parameter is not valid for OPTion (F).

KEYLENF

Full-word binary integer, posted by the exit. Generally set during OPTion (O) processing by the exit program and should contain the whole key length for the file.

KEYLENR

Full-word binary integer, posted by the exit. Generally set during OPTions (G) and (E) processing by the exit program and should contain the actual key length for a direct read.

ERRTEXT

Full-word binary integer, posted by the exit. Should contain the absolute starting address of a message. FOCUS displays this message if the RC parameter returned by the exit is non-zero and for OPTions (E), (G), and (S).

ERRTEXTL

Full-word binary integer, posted by the exit. It should contain the length of the above ERRTEXT message.

INDEX

Full-word binary integer, posted by GNTINT.

This option contains the index # by which to access the file.

- 0 Primary key in Master File in Master -- KEY-DKEY
- 1, 2, e Secondary indexes in the Master File
- tc KEY1-DKEY1 or KEY2-DKEY2, and so on, and INDEX=I

RESERVE1

Full-word binary integer, reserved for future use.

User Exits for Non-FOCUS Data Sources

USERID

Full-word binary integer, posted by GNTINT. This userid will only be present if found in the central site security environment.

RESERVE2

Full-word binary integer, reserved for future use.

Example

Sample Assembler DSECT

```
GETPRVPL DS    0F
NCHAR@   DS    A
DDN@     DS    A
ABUF@    DS    A
RC@      DS    A
KEY@     DS    A
OPT@     DS    A
CONTEXT@ DS    A
          TITLE 'GETPRV CONTEXT'
CONTEXTD DSECT
EYE      DS    CL8   eye catcher literal
PFMCB@   DS    A     handle
PFACB@   DS    A     file open handle
PFRPL@   DS    A     file retrieval handle
KEYLEN_F DS    F     key length for file
KEYLEN_R DS    F     key length for this request
RETTEXT@ DS    A     pointer to returned message
LRETTEXT DS    F     length of returned message
INDEX    DS    F     index for file access - 0 primary 1... secondary
          DS    A     reserved
USERID   DS    CL8
          DS    2F   reserved
          SPACE 1
```

Example Sample C Declaration Required For Invoking FFUN Function

```

/*
control block for additional info
*/
typedef struct getprv_inf_s {
char      eye[8];      /* I: eye catcher "PRIVATE "*/
Pvoid     pfmcb;      /* o: p' to handle forgetprv  */
/*      set up by user at first option O */
/*      I: p' to handle for getprv  */
/*      passed to user by all other calls*/
Pvoid     pfacb;      /* o: p' to handle for file */
/*      set up by user at option O      */
/*      i: p' to handle for file */
/*      passed to user at option C,R,E,G,S */
Pvoid     pfrpl;      /* o: p' to handle for request */
/*      set up by user at option R      */
/*      i: p' to handle for request */
/*      passed to user at option E,G,S */
long      keylen_fil; /* I: key length (whole) for the file . */
/*      used at option O . */
long      keylen_req; /* I: key length for the direct read request*/
/*      used at direct read options G,E */
char      *rettext;  /* O: a'native db error msg text */
long      lrettext;  /* O: l'native db error msg text */

long      index;     /* I:index # by which to access file :
                    = 0      - primary key
                    in master - KEY|DKEY
                    = 1,2,... - secondary indexes
                    in master - KEY1|DKEY1 or KEY2|DKEY2 ...
                    and INDEX=I */

long      res1[1];   /* reserved */
char      userid[8]; /* user id */
long      res2[2];   /* reserved */
} getprv_inf_t;
/*

*/
typedef void FFUN getprv_t (
long      *nchar     /* out: length of data record read. =0 */
/*      if eof or no rec found      {*/
/*      used in all read options S,E,G */
, char    *ddn       /* in : ddname to read      */
/*      used in all options except F */
, char    **abuf     /* out: a' buffer */
/*      used in all read options S,E,G */
, long    *rc        /* out" return code. = 0 if ok */
/*      used in all options          */
, char    *key       /* in: key value for read */
/*      used in read options E,G     */

```

User Exits for Non-FOCUS Data Sources

```
,char      *opt      /* in: read option : */
                /* S sequential read */
                /* G      GE read */
                /* E      EQ read */
                /* control options */
                /* O open file */
                /* R open request (position)*/
                /* C close file */
                /* F fin of focus */
,getprv_inf_t *      /* in/out other info .see above */
);
#endif
```

User-coded Data Access Modules

You can write a user routine to provide records to the FOCUS report writer from any non-standard data source not directly describable to FOCUS using the Master File. FOCUS treats the record, which can come from any data source, exactly as if it came from a FOCUS data source. The user routine must be coded as a subroutine in FORTRAN, COBOL, BAL, or PL/I, and must pass data to the FOCUS calling program through arguments in the subroutine.

The user program is loaded automatically by the report writer and is identified by the file suffix, in the Master File. If, for example, the Master File contains:

```
FILE = ABC, SUFFIX = MYREAD
```

FOCUS will load and call the program MYREAD to obtain data whenever a TABLE, TABLEF, MATCH, or GRAPH command is issued against file ABC.

The record returned to FOCUS must correspond to a segment instance in the Master File. The layout of the fields must correspond to the ACTUAL formats specified.

The user program is responsible for determining which segment to pass to FOCUS if the Master File has more than one segment. FOCUS will traverse the hierarchy in a top-to-bottom, left-to-right sequence; if the user routine can anticipate which segment FOCUS expects, the number of rejected segments will decrease and retrieval efficiency will improve.

In FORTRAN, the subroutine MYREAD would be coded as follows:

```
SUBROUTINE MYREAD (LCHAR, BUF, OFFSET, RECTYP, NERRX, CSEG, REGI, NFIND, MATCH,
IGNOR, NUMFLD, NUMLEV, CVT)
```

where:

LCHAR

(4 byte integer) If LCHAR > 0, LCHAR is the number of characters in the record passed back to FOCUS. If LCHAR = 0, the user routine is telling FOCUS that an end-of-file has been encountered and the retrieval is complete. If LCHAR = -N, the user routine is telling FOCUS that the buffer contains an error message of length N, to be printed out, and that the user routine should not be called again. LCHAR = -1 is reserved for Information Builders use.

BUF

This parameter is a 4096 byte buffer provided by FOCUS to receive the record from the user routine.

OFFSET

(4-byte integer) If the user routine puts data in BUF, OFFSET should be set to 0 each time the user routine is called. If the user routine provides its own buffer or buffers, OFFSET is the address of the user's buffer minus the address of BUF. A utility called IADDR is provided to compute an address. In FORTRAN, for example, one could code:

```
OFFSET = IADDR (MYBUF) - IADDR (BUF)
```

RECTYP

(4-byte integer) The number of the FOCUS segment corresponding to the record being presented to FOCUS, set by the routine. These numbers correspond to either the list obtained by issuing '? FILE filename' or the field SEGNO resulting from a CHECK FILE filename HOLD.

NERRX

(4 byte integer) Flag set by FOCUS. If NERRX < 0, FOCUS is directing the user routine to shut down (for example, close all files) and not provide any more records. On return, FOCUS will not call the subroutine again.

CSEGX REGI NFIND MATCH

Reserved for use by Information Builders.

IGNOR

(4-byte integer) FOCUS will reject any segment whose number (see RECTYP) is greater than or equal to IGNOR. IGNOR is set by FOCUS, based on the segments referenced in the request, but may be reset by the user routine.

NUMFLD NUMLEV

Reserved for Information Builders. Use CVT.

On CMS, FOCUS looks for MYREAD TEXT on any accessed CMS minidisk. If MYREAD TEXT is not found, FOCUS next looks for member MYREAD within all TXTLIB files currently pointed at by a GLOBAL TXTLIB command.

On TSO, the MYREAD object code, and any other routine that it calls, must be link-edited into a load module in the load library with member name MYREAD. This load library can be allocated as ddname USERLIB or concatenated with the STEPLIB program library.

Re-Entrant VSAM Compression Exit: ZCOMP1

The re-entrant ZCOMP1 exit was designed as an exit users can supply to decrypt coded fields, expand compressed records, and accomplish other specified data manipulations. The ZCOMP1 user exit can access all files readable by the GNTINT Interface and it is the user's responsibility to write and maintain that exit. It works with compressed VSAM and sequential files. It requires an initial call to ZCOMP0 for initial housekeeping and a USERWD parameter to anchor storage.

There are no special Master File requirements. SUFFIX can equal VSAM (for KSDS or ESDS files), FIX (for sequential files), or PRIVATE (for file access through the GETPRV user exit).

Linking ZCOMP1

After you write a ZCOMP1 user exit, you link it with VVSET by using either the GENFSAM EXEC (for VM/ESA), or the GENFSAM JCL (for OS/390) found in the FOCCTL.DATA data set. Note that GENFSAM is designed to link in both ZCOMP1 and GETPRV user exits at the same time. If you only implement one of them, the VM EXEC generates the correct linkage to the routine required, whereas in the OS/390 JCL, you must comment out the INCLUDE OBJECT statements in the GENFSAM member. ZCOMP1 can be linked as re-entrant if you plan to use the USERWD parameter.

What Happens When You Use ZCOMP1

The GNTINT Interface reads the record for the allocated data source. Upon a successful read, GNTINT calls ZCOMP0, if it exists, with the parameters listed below so that initial housekeeping can be performed. All subsequent calls are to ZCOMP1 with the same parameter list.

The ZCOMP1 exit is responsible for determining what to do with the parameter information it receives. The DDNAME can be used to determine whether the associated data source needs to be decompressed or not. If not, the user exit typically moves A(IRECLEN) to A(ORECLEN) and A(A(IREC)) to A(A(OREC)) and returns to GNTINT with a zero A(STATCODE). If decompression or other processing is required, it is the responsibility of the user exit to do so.

After the user exit completes its processing, it should return with either the A(ORECLEN), A(A(OREC)) and a zero status code or with a non-zero status code which gives the following message:

```
(FOC1150) ZCOMP DECOMPRESS ERROR: status
```

Note: This error terminates a TABLE request.

ZCOMP1 Parameter List

Parameter	Description	Length and Format
A(STATCODE) *	Pointer to fullword binary status code	4-byte integer
A(DDNAME)	Pointer to 8 byte file name in use	8-byte character
A(USERID)	Reserved for future use	8-byte character
A(IRECLLEN)	Pointer to length of original record	4-byte integer
A(A(IREC))	Pointer to pointer to original record	4-byte integer
A(ORECLLEN) *	Pointer to length of revised record	4-byte integer
A(A(OREC)) *	Pointer to pointer to revised record	4-byte integer
A(USERWD) **	Pointer to fullword	4-byte integer

* The user supplies these parameters.

** This parameter can be used to anchor user storage for re-entrant processing.

Note: Never modify the primary pointers - only the pointers or values that they point to.

- Note that upon entry to ZCOMP1, the ORECLLEN and OREC parameters are NULL. It is the responsibility of the user to fill these in correctly.
- While processing the FOCUS FIN command, a call is placed to the ZCOMP2 entry point, which provides the user with the ability to do any other global cleanup required.
- The parameters returned by ZCOMP1 are not validated. It is the responsibility of the user routine to ensure that valid addresses and lengths are returned to FOCUS from ZCOMP1.
- Unpredictable results occur if incorrect parameters are passed back from the routine.

Index

A

Access Files
 For GNTINT Interface, B-3

Accessing fields in combined files
 FIND, 3-8

accessing VSAM files
 from CMS, 3-2
 from MVS, 3-2

ACTUAL
 Field length attribute, 2-3
 GROUP attribute, 2-3

Adjust current position in file
 REPOSITION, 3-8

AIX name
 associated Path names, 2-17

ALIAS
 KEY, 2-2
 MAPFIELD, 2-11

ALIAS=KEY
 GROUP attribute, 2-2

allocating VSAM files
 VSAM/SU mode, 3-3

alternate indexes
 mapping to base cluster, 3-2

Alternate indexes
 Automatically selecting, 2-15
 VSAM files, 2-15

Attributes
 Master File
 GROUP, 2-4

B

Buffers
 VSAM, 2-14

BUFND
 Establish data buffers, 2-14

BUFNI
 Establish index buffers, 2-14

C

Check status of XE "XMISAMT FOCEXEC"
 VSAM/SU FOCUS Database Server
 XMISAMT FOCEXEC, 3-4

Check status of VSAM/SU FOCUS Database
 Server, 3-4

clusters. *See* VSAM data sets

COBOL FD Translator
 Converting VSAM file descriptions, 2-1

COMBINE
 Syntax, 3-7

COMBINE command, 3-7

Combining
 RECTYPE and OCCURS, 2-17

Command syntax
 COMBINE, 3-7
 FIND function, 3-8
 LOOKUP, 3-6
 REPEAT, 3-10
 REPOSITION, 3-8

Converting VSAM file descriptions
 COBOL FD Translator, 2-1

D

- Data
 - Access
 - Via a user coded routine, B-8
- Decimal fields
 - Packed
 - External data sources, 2-3
- DEFINE function. *See* IDCAMS facility. *See* Creating VSAM files
- DEFINE parameters
 - IMBED, 3-2
 - NOREPLICATE, 3-2
 - SHAREOPTIONS, 3-2
- Describing Alternate Index and Path
 - DEFINE, 3-2
- Describing data relationships
 - Unrelated records, 2-7
- Describing field formats
 - ACTUAL, 2-3
 - USAGE, 2-3
- Describing files with different descendent segments, 2-17
- Describing files with unrelated records
 - DUMMY segname, 2-9
- Describing groups
 - GROUP attribute, 2-2
- Describing records with repeating groups
 - OCCURS, 2-10
- DUMMY SEGNAME
 - With ISAM and VSAM files, 2-9

E

- Exits
 - GNTINT Interface, B-1
 - ZCOMP1, B-10

F

- Field formats
 - ACTUAL, 2-3
 - USAGE, 2-3
- Field lengths
 - ACTUAL, 2-3
 - Describing, 2-3
 - USAGE, 2-3
- Field names
 - MAPFIELD, 2-12
- Fields
 - Packed decimal
 - External data sources, 2-3
- FIELDTYPE=I
 - Master File Attribute, 2-16
- File(s)
 - VSAM files, key-sequenced, 2-2
 - VSAM, key-sequenced
 - Positional records, 2-5
- FIND
 - Syntax, 3-8
- Finding AIX names, 2-17
- Finding alternate index names, 2-16
- Finding path names, 2-16
- Finding Path names, 2-17

G

- GNTINT
 - Access File, B-3
 - Calling sequence, B-3
 - Private user exit, B-1
 - Work area control block, B-5
 - ZCOMP1 user exit, B-10

Group
 Keys
 Describing unrelated records, 2-7
 Keys for screening, 2-4
 Keys in ISAM, VSAM files
 Determining lengths, 2-5

GROUP
 Master File Attribute, 2-2

GROUP attribute
 ALIAS, 2-2

GROUP attributes
 ACTUAL and USAGE, 2-3

GROUP declarations
 coding, 2-2
 Field lengths, 2-3

Group keys in VSAM files
 Describing unrelated records, 2-2

H

Handling multiple segment occurrences
 REPEAT, 3-9

I

IDCAMS functions, 3-2
 copy, 3-2
 create, 3-2
 delete, 3-2
 list, 3-2
 verify, 3-2

IMBED. *See* DEFINE

ISAM data sources
 Simple, 2-1
 SUFFIX attribute, 2-2

ISAM files
 Complex
 Describing positionally-related records, 2-5
 Describing unrelated records, 2-7
 See ISAM data sources, 2-1

ISAM Files
 Unrelated records, 2-7

K

Key fields
 describing group keys, 2-4
 Group keys in files with unrelated records, 2-9

Key fields in VSAM files
 Describing groups, 2-2

L

Loading data. *See* MODIFY

LOOKUP
 Syntax, 3-6

LOOKUP function, 3-6

Looping through files
 REPEAT, 3-9

M

MAPFIELD
 ALIAS, 2-12
 VSAM repeating groups, 2-11

MAPVALUE
 supplies MAPFIELD value, 2-12

Master File
 For GNTINT Interface, B-3

Index

Master File Attributes

- ACTUAL, 2-3
- Describing alternate indexes, 2-16
- FIELDTYPE=I, 2-16
- GROUP, 2-2
- MAPFIELD, 2-11
- MAPVALUE, 2-12
- OCCURS, 2-10
- RECTYPE, 2-7, 2-10
- SEGNAME, 2-2, 2-7
- SEGTYPE, 2-2
- SUFFIX, 2-2
- USAGE, 2-3

MODIFY procedure

- Loading VSAM data, 3-2

Modifying unrelated records

- RECTYPE, 3-18

Modifying variable-length segments

- OCCURS, 3-11

Module

- User coded
 - For data access, B-8

N

NOREPLICATE. *See* DEFINE

O

OCCURS

- Combining with RECTYPE, 2-17

OCCURS=n processing, 3-15

OCCURS=VARIABLE

- Segments, 2-19

OCCURS=VARIABLE processing, 3-11

P

Packed decimal fields

- External data sources, 2-3

PARENT

- Master File attribute, 2-2

Path names

- for given AIX name, 2-17

Positionally-related records

- Describing VSAM files, 2-5

PRIVATE

- Suffix for GNTINT Interface, B-3

Q

Query commands

- VSAM buffer, 2-15

R

Reading

- external files
 - User-written procedures, 2-19

Record selection

- MATCH and NEXT, 3-4

RECTYPE

- MAPFIELD, 2-11
- Repeating groups, 2-10

RECTYPE attribute

- VSAM files. *See also* Master File Attributes

REPEAT, 3-9

- Syntax, 3-10

Repeating non-key fields

- GROUP attributes, 2-3

REPOSITION

- Syntax, 3-8

Retrieving data from cross-referenced files

- MODIFY LOOKUP, 3-6

S

Screening

- GROUP key values, 2-4

Segment name
 VSAM files, 2-2

SEGNAME
 Master File attribute, 2-2

SEGTYPE
 VSAM files, 2-2

Sequential files
 Positionally-related records, 2-5

SET
 BUFND, 2-14
 BUFNI, 2-14

SHAREOPTION values, 3-3

SHAREOPTIONS. *See* DEFINE

Subroutines
 User coded
 For data access, B-8

SUFFIX
 ISAM or VSAM, 2-2
 Master File attribute, 2-2
 PRIVATE, B-3

T

Testing
 Group keys, 2-4

U

Unrelated records
 In ISAM and VSAM files, 2-7

Updating multiple files
 COMBINE, 3-7

USAGE
 Field length attribute, 2-3
 GROUP attribute, 2-3

User
 Data access routine, B-8

User exit
 GNTINT Interface, B-1
 ZCOMP1, B-10

V

Variable-length segments
 Modifying, 3-11

Variably occurring segments
 describing with OCCURS, 2-19

VSAM catalog
 Defining with IDCAMS, 3-2

VSAM Cluster concepts
 Alternate index and path names, 2-16

VSAM data sets, 3-2

VSAM data sources
 See also ISAM files, 2-1
 Simple, 2-1
 SUFFIX attribute, 2-2

VSAM file attributes
 SEGNAME, 2-2
 SUFFIX, 2-2
 ISAM or VSAM, 2-2

VSAM file descriptions
 Conversion for FOCUS, 2-1

VSAM files, 2-5
 allocating MF name to CLUSTER, 2-1
 Complex

Describing unrelated records, 2-7
 Positionally-related records, 2-5

Data and index buffers, 2-14

GNTINT Interface

User exit, B-1
 ZCOMP1, B-10

Positionally-related records, 2-5

Query for buffer settings, 2-15

Repeating groups, 2-14

Index buffers, 2-14

MAPFIELD, 2-11, 2-14

RECTYPE, 2-10, 2-11

Index

VSAM files continued

See VSAM data sources, 2-1

Segment name attribute, 2-2

SEGTYPE, 2-2

VSAM Files

RECTYPE attribute, 2-7

VSAM files, key-sequenced, 2-1, 2-5

VSAM Files, key-sequenced

Unrelated records, 2-7

allocating files, 3-3

VSAM Group keys, 2-1

VSAM/SU mode

X

XMISAMT FOCEXEC, 3-4

Z

ZCOMP1, B-10

Reader Comments

In an ongoing effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes your suggestions regarding this manual.

Please use this form to relay suggestions for improving this publication or to alert us to needed corrections. Please identify specific pages where applicable. Send comments to:

Corporate Publications
Attn: Manager of Documentation Services
Information Builders
Two Penn Plaza
New York, NY 10121-2898

or FAX this page to (212) 967-6406, or call **Jay Patrick** at (212) 736-6250, x**3708**.

Name: _____

Company: _____

Address: _____

Telephone: _____ Date: _____

Comments:

Reader Comments
