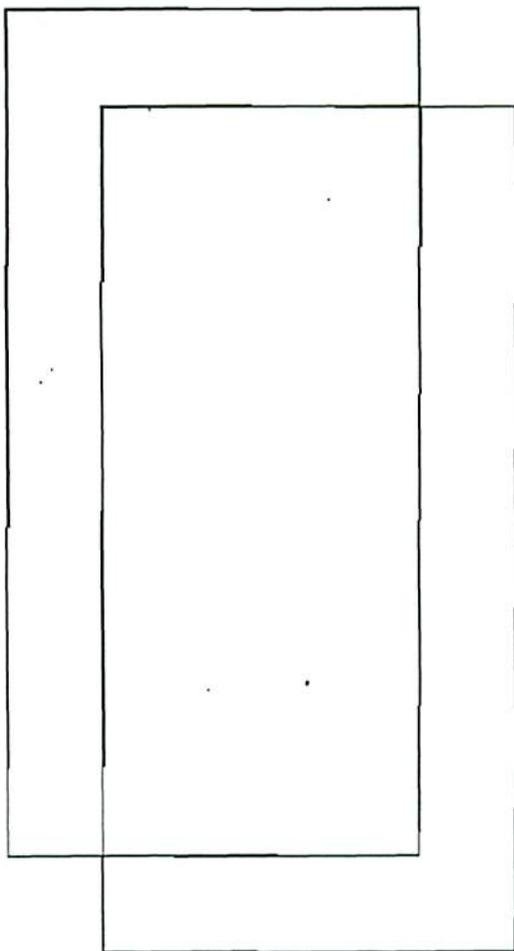


ESTTA Tracking number: **ESTTA321549**

Filing date: **12/11/2009**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE TRADEMARK TRIAL AND APPEAL BOARD

Proceeding	91179897
Party	Plaintiff Information Builders, Inc.
Correspondence Address	Howard F. Mandelbaum Levine & Mandelbaum 222 Bloomingdale Road, Suite 203 White Plains, NY 10605 UNITED STATES mail@levman.com
Submission	Testimony For Plaintiff
Filer's Name	Howard F Mandelbaum
Filer's e-mail	mail@levman.com
Signature	/Howard F Mandelbaum/
Date	12/11/2009
Attachments	PX3 of 87-Part 1.pdf (45 pages)(592285 bytes) PX3 of 87-Part 2.pdf (45 pages)(537766 bytes) PX3 of 87-Part 3.pdf (45 pages)(563320 bytes) PX3 of 87-Part 4.pdf (45 pages)(429433 bytes) PX3 of 87-Part 5.pdf (45 pages)(479587 bytes) PX3 of 87-Part 6.pdf (35 pages)(494100 bytes)



F O C U S

USER'S MANUAL



INFORMATION BUILDERS INC.
254 WEST 31st STREET
NEW YORK, N.Y. 10001
(212) 736-4433

TABLE OF CONTENTS

INTRODUCTION.3-01
CONVENTIONS3-02
Diagram Conventions3-02
Related Segments3-02
Cross-references3-04
Instances of data.3-05
Creating FOCUS File Descriptions.3-06
Typing Conventions3-06
Checking a File Description3-08
Changing a FOCUS File Description3-09
Storing FOCUS File Descriptions3-10
Hidden Source File3-11
FILE STRUCTURES.3-12
Single Path File Structure.3-13
File Attributes3-13
Segment Attributes3-13
Field Attributes3-15
Multiple-Path File Structure3-17
Unique Segment.3-18
Cross-References Between Files3-22
Preparing a Segment for Sharing - FIELDTYPE=I3-26
Decoding.3-27
Cross-Referencing Multiple Instances3-29
Sharing Linked Segments.3-33
Segment type KLU3-35
Hierarchies of Linked Segments3-36
SPECIAL TOPICS3-38
Re-naming Cross-Referenced Fields3-39
Complex File Structures.3-41
Multiple Parents3-41
Recursive Reuse of a Segment3-42
Combining FOCUS Files at Run Time3-44
The USE Command3-44
Partitioned Data Files3-46
Extended Size Files3-47
Data Base Administrator Tools.3-48
The Data Administrator Package (DAP)3-48

TABLE OF CONTENTS (cont'd)

FILE DESCRIPTION ATTRIBUTES.3-50
FILENAME.3-51
SUFFIX3-52
SEGNAME3-53
PARENT3-54
SEGTYPE3-55
CRKEY.3-57
CRFILENAME3-58
FIELDNAME3-59
ALIAS.3-60
USAGE FORMAT3-61
Dates3-64
Date Translation.3-64
 REFERENCE GUIDE - ATTRIBUTES3-65

FOCUS Files

A data storage system on a computer is like a filing cabinet in which the various draws hold folders of information which can be retrieved, and the items in the folders cross-reference other draws where other related information is available. The organization of the material in the draws should be logical, and reflect the way the information is both collected, and used. This logical organization assists the user in storing the information as it is gathered, and finding it when it is needed.

The basic building block of a FOCUS file is the individual item of data, or field. Each field has a set of associated attributes such as the name by which the field is to be known, the type of data values stored in each occurrence of the field, i.e. computational numbers, alphanumeric characters, etc.

A group of data fields which are related to one another, usually in a one-to-one fashion compose a record segment, or segment for short. For example, a persons full name, date of birth, and address might logically be grouped on the same segment. The segments are likewise given names to identify them, and they also have attributes which serve to relate one segment to another. In simple files only a small subset of the field and segment attributes have to be specified to describe the way the information is related. In more elaborate files a larger number of attributes are used.

The attributes used to describe a particular collection of fields compose the file description. In a given application several files may be needed, and they may share common information and cross-reference each other. The descriptive process is open ended, and new ways to look at the collected data may be constructed when needed. These new file 'views' may combine existing segments with new data to be collected.

This section of the User's Manual is devoted to describing the attributes of FOCUS files, how they are typed and checked, and some of the concepts of file design which are useful in a range of applications from the simplest to most complex.

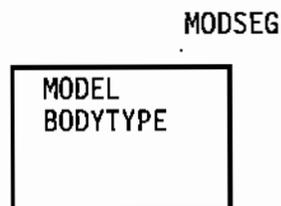
Diagram Conventions

A convenient way to illustrate how various data fields and the segments they reside on relate to one another in a file description is to draw a diagram with boxes representing the segments, and arrows between the boxes showing the type of connection. Inside the boxes some of the data fields, usually only the 'key' fields needed to identify different instances of the data will be given.

For instance, if a set of fields representing information on imported cars are:

MODEL
BODYTYPE
SEATS
RETAIL COST
ENGINECODE

and they are grouped together on a segment named MODSEG, then in a diagram this will be represented as:



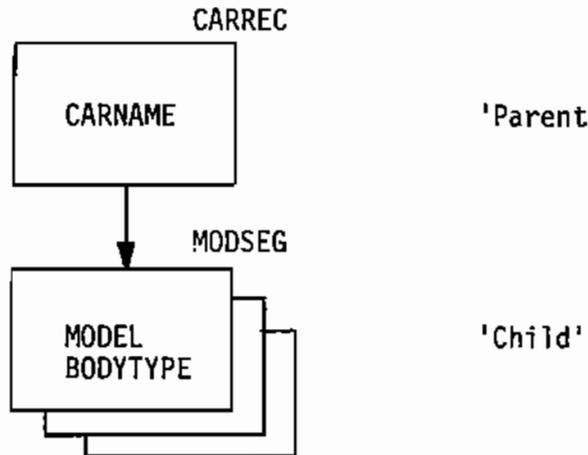
Related Segments:

When segments are related to each other, in a hierarchical fashion such that one segment is common to many descendent segments so that a parent to child bond between the information exists then a solid line will be drawn from the parent segment to the child segment. An arrow at the end of the line indicates the direction of the relationship. In a FOCUS file we can generally travel from a child segment, to a parent segment, as well as parent to child, hence there may be arrows on both ends of a solid line.

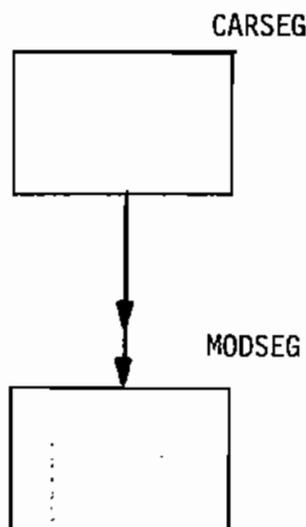
Segments which can occur a multiple number of times relative to a given parent will be denoted by projected boxes rather than a single box. This means that the segments have a 'one to many' relationship. e.g. a parent with several children.

Related Segments: (cont'd)

For instance, MODSEG, containing information about a car model, is dependent on the segment which identifies the car. Since there can be many models for a given car the relationship can be diagrammed as:



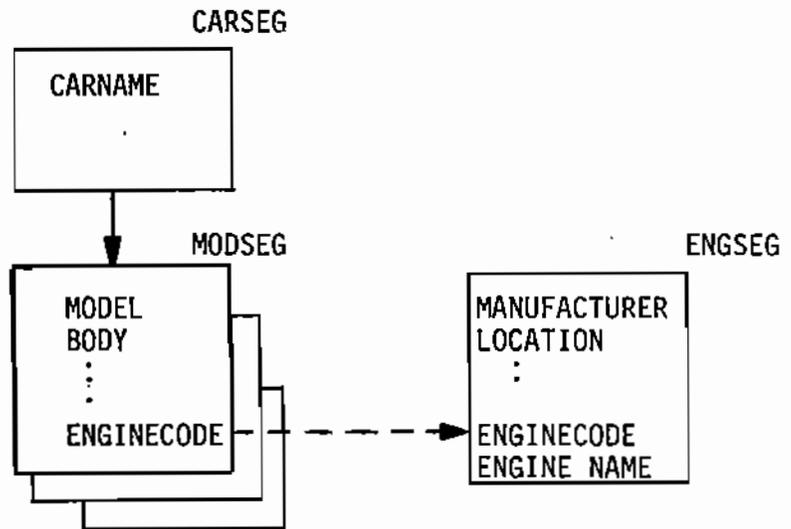
In some cases a more concise notation for a one to many relation will be used; this is a line with two arrows on the same end, e.g. $\longrightarrow\longrightarrow$



Cross-references:

A solid line between segments means that a structural relationship exists between the segments. For instance, a MODSEG with the names of car models may be meaningless without its parent to help to fully identify it. For example, a data instance of MODEL=128 2 DOOR DIX, is not meaningful unless we know that its parent was CARNAME=FIAT. Another type of data connection is a cross-reference. This occurs when the same data item occurs in another segment, perhaps in an entirely different context. A dotted line will be used to connect segments for which a cross-reference exists. An arrow on the line is the direction of cross-reference. Cross-references may be one to one, or one to many and projected boxes will be used in a one to many case. (or a double arrow on one end of the dotted,----->>).

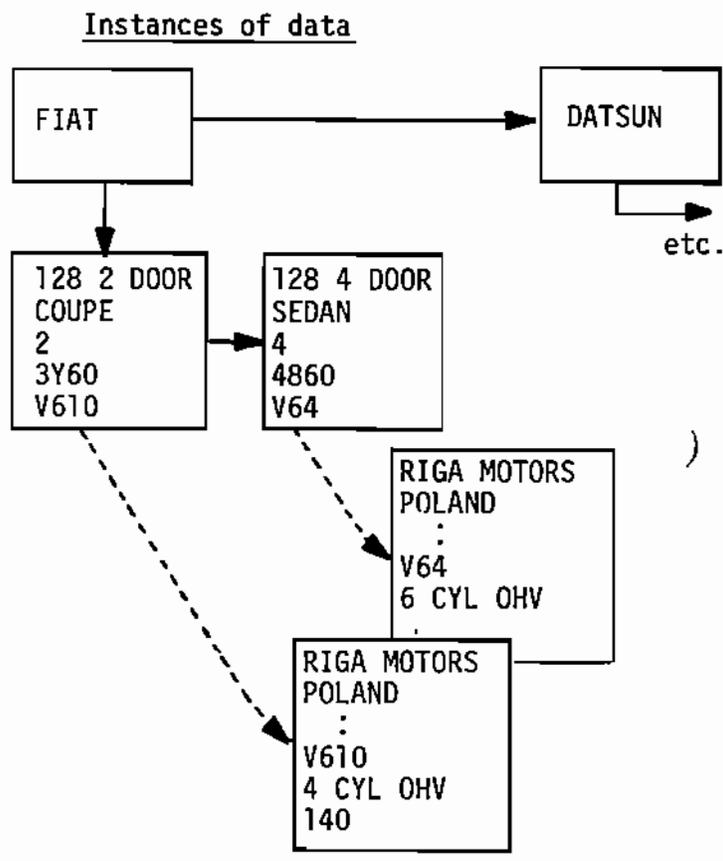
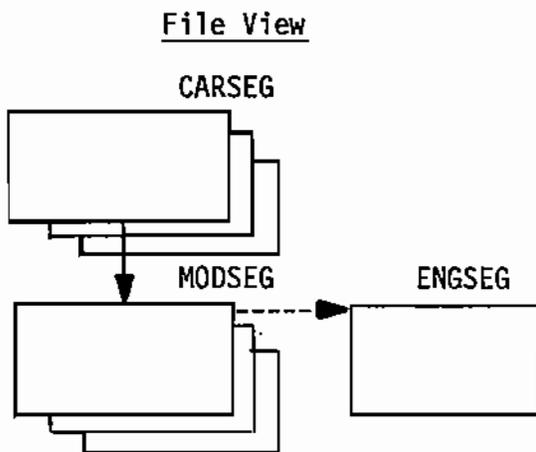
An example of a simple cross-reference is illustrated if the ENGINE-CODE field in MODSEG also occurs in another segment named ENGSEG which contains descriptive information of the name of the engine, its horsepower, etc. The file description for this case would be diagrammed as:



Instances of Data:

Another type of diagram useful for illustrating how data might actually occur in a file is the 'instance' diagram. Each box in an instance diagram represents one occurrence, or instance, of a segment. An arrow connects one instance to another.

Example:



Creating FOCUS File Descriptions

The description of a FOCUS file is typed into a CMS file whose file-type is MASTER. The CMS filename becomes the name by which the file is also known to FOCUS. Hence, a FOCUS file named FCARS must have its source description stored in a CMS file named FCARS MASTER. The CMS 'Editor' is used to create this source description, and also to make any changes in it.

The attributes which comprise the source description are typed in a free format with commas separating one attribute from another. For example a line in the source description might be typed as:

FIELD=RETAIL COST, ALIAS=RCOST

attribute value comma separator

Typing Conventions:

There are three classes of attributes used to describe a FOCUS file. These are File attributes, Segment attributes, and Field attributes. The attributes describing the file, segments, and fields in a FOCUS file can be typed one after the other in a free form comma delimited format. At the end of the description of each element a comma followed by a dollar sign '\$' is placed to signify the end for that element. For example,

```
FIELDNAME=DEALER-COST, ALIAS=DCOST, USAGE=D8, $  
FIELDNAME=RETAIL-COST, ALIAS=RCOST, USAGE=D8, $
```

The rules for free format data when entering file descriptions are:

- . Attribute names and data values may begin in any column of the line.
- . Leading blank spaces in front of any name or value are ignored.
- . Commas separate one set from another.
- . Attribute names may be identified by their full name, alias name, or shortest unique truncation.

i.e. FIELDNAME=, or FIELD=

Typing Conventions: (cont'd)

- . The end of a set of elements is denoted by a comma followed by a dollar sign. Hence, the description for a field can occupy as many lines as needed. i.e.

```
FIELDNAME=CAR
  ALIAS=CR, USAGE=A18, $
```

The meaning of the various attributes are each described in a section of this manual.

Illustrated below are comma separated data for the file attributes of FILENAME and SUFFIX.

```
FILENAME=FCAR, SUFFIX=FOC, $ ← note terminator
```

Directly after the file attributes comes the description of the first segment of data. It might be typed as:

```
SEGNAME=ORIGIN, SEGTYPE=S, $ ← note terminator
```

Directly after the segment attributes are the field attributes for each field on the segment. At the conclusion of a list for one field the terminator is typed. For instance two data fields may be described as:

```
FIELD=CARNAME, ALIAS=CAR, USAGE=A18, $ ← note terminator
FIELD=CARCODE, ALIAS=CCODE,
  USAGE=A4, $ ← note two lines used
```

Checking a File Description:

It is easy to correct a file description using the CMS Editor facilities, but it is more difficult to determine if there are any typing errors, or violation of rules. Hence, a checking procedure is used to assist the file designer.

Enter FOCUS and at the command level type:

```
EX TESTFILE
```

The procedure will ask for the name of the file description to be checked. If there are any errors found they will be listed on the terminal. These errors must be corrected before any attempt is made to enter data for the file.

Remember, the file description must be stored in a CMS file whose file-type is 'MASTER'.

Changing a FOCUS File Description:

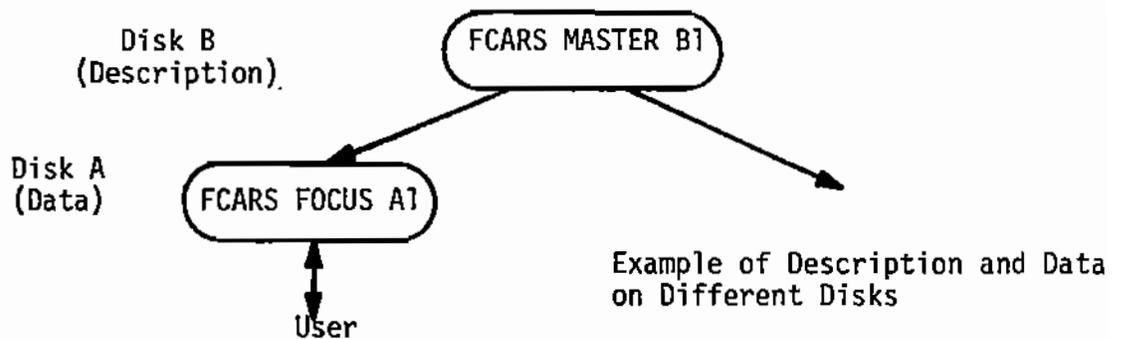
Once data has been entered for a file it is no longer possible to make arbitrary changes to the file description. Some changes are entirely harmless and can be made at any time, others are prohibited unless the data is re-entered or the file rebuilt. A few others can be made if corresponding changes are made in several places.

The section in this manual describing each attribute provides a guide for the degree of change allowed to the values for that attribute.

When changes can be made the CMS file containing the 'source' lines of the file description and any other related files are changed using the CMS Editor facilities. The check-out procedure, EX TESTFILE, should be executed after any change.

Storing FOCUS File Descriptions

The description of a FOCUS file and all files it cross-references must be available whenever a reference to the file is made. For instance when a report is requested. Generally the description and the data reside on the same disk, the 'A' disk, but the descriptions do not have to be on the same disk as the data, and many users may share the same set of descriptions. i.e.



When data is entered for a new file it is stored in a CMS file whose default filetype is FOCUS. Note in the above example that the file named FCARS has its description in FCARS MASTER, and the data is in FCARS FOCUS.

When these defaults are not changed then no further action is required on the users part. FOCUS can be entered and reports prepared, or data changed without any FILEDEF, or other communication. When the default filetype is renamed, or the data does not reside on the 'A' disk then the FOCUS USE command provides the full name of the data file to use. For instance, if the data is on the 'B' disk then before it can be accessed the following lines must be typed when FOCUS is entered.

```
USE
FCARS FOCUS B1
END
```

See the section 'The USE Command' for further details of its usage.

Hidden Source File:

The various MASTER files do not have to be available in their 'source' form. The data base administrator may collect them in one (or more) FOCUS dictionaries. From this central dictionary the descriptions are available for all purposes, but cannot be directly accessed themselves by users of the data. See the section 'Data Base Administrator Package - DAP' for details on this subject.

BASIC ATTRIBUTES USED FOR DESCRIBING

SINGLE PATH HIERARCHICAL FILES

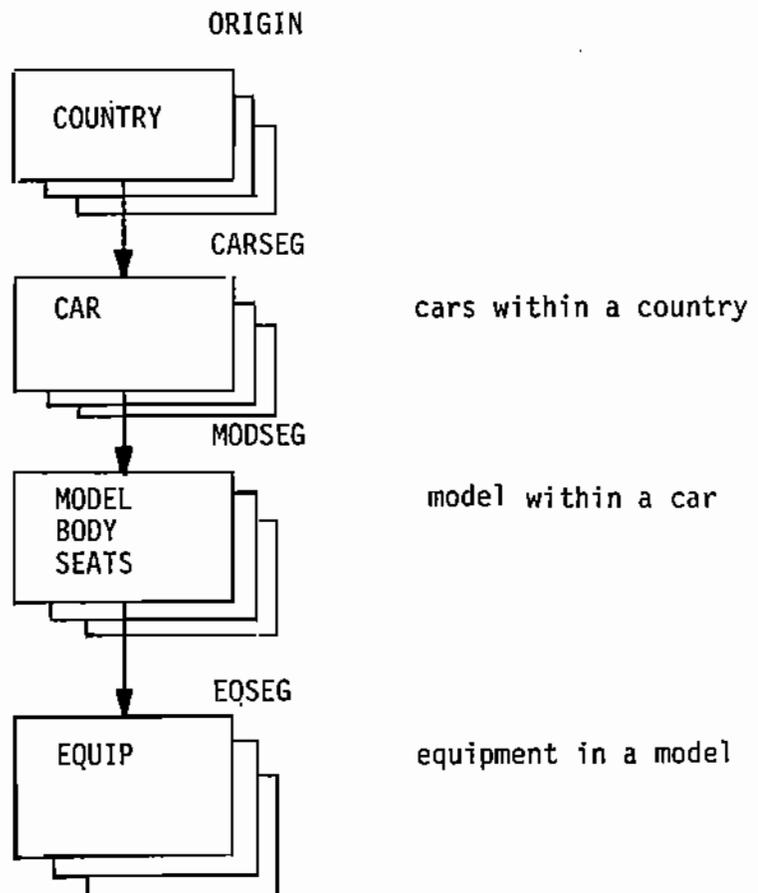
- * File Attributes
- * Segment Attributes
- * Field Attributes

Single Path File Structure

One of the most common forms of organization of a collection of data items is into a hierarchical structure. As the name implies there are levels in a hierarchy. The topmost level has information common to all lower levels. Each successive level adds greater particularity. For example, in the imported cars case the collection of data items are:

<u>Field</u>	<u>Meaning</u>
COUNTRY	Country of origin
CAR	Name of car
MODEL	Name of model of car
BODY	Bodytype of model
SEATS	Number of seats in model
EQUIP	Standard equipment in model

One method of organization of these items would be a four level hierarchy. A diagram of this file view would be:



Single Path File Structure (cont'd)

The reason this is called a single path hierarchy is because each segment has only one type of descendent. Hence, in the diagram there is only one path from top-to-bottom.

File Attributes:

The file attributes needed to describe any FOCUS file, not just single path hierarchies are:

<u>ATTRIBUTE NAME</u>	<u>ATTRIBUTE ALIAS</u>	<u>LENGTH</u>	<u>MEANING</u>
FILENAME	FILE	1-8 CHARS	NAME OF FOCUS FILE
SUFFIX	FILESUFFIX	3 CHARS	MUST BE LETTERS 'FOC'

(See section Description of File Attributes for other details).

Segment Attributes:

Three pieces of information are used to describe each segment in a hierarchical file. These are:

<u>ATTRIBUTE NAME</u>	<u>ALIAS</u>	<u>LENGTH</u>	<u>MEANING</u>
SEGNAME	SEGMENT	1-8 CHARS	NAME OF SEGMENT
PARENT	PARENT	1-8 CHARS	NAME OF PARENT TO THIS SEGMENT
SEGTYPE	SEGTYPE	1-4 CHARS	TYPE OF SEGMENT SEQUENCING

The hierarchical structure of the file is controlled through the value assigned to the attribute PARENT. In a single path situation each segment names its immediate predecessor as its parent.

Field Attributes:

Three pieces of information are needed about every data field regardless of file organization. These are:

<u>ATTRIBUTE NAME</u>	<u>ALIAS</u>	<u>LENGTH</u>	<u>MEANING</u>
FIELDNAME	FIELD	1-12 CHARS	Name assigned to data item
ALIAS	SYNONYM	1-12 CHARS	Alternate names assigned to identify item
USAGE FORMAT	FORMAT	2-8 CHARS	Type, length, and edit options of data values

The FCARS file could be typed in FCARS MASTER as:

Example: Single Path Hierarchical File

```

FILENAME=FCAR, SUFFIX=FOC, $
  SEGNAME=ORIGIN, SEGTYPE=S, $
    FIELD=COUNTRY, ALIAS=CTY, USAGE=A20, $
  SEGNAME=CARSEG, SEGTYPE=S, PARENT=ORIGIN, $
    FIELD=CARNAME, ALIAS=CAR, USAGE=A14, $
  SEGNAME=MODSEG, SEGTYPE=S2, PARENT=CARSEG, $
    FIELD=MODEL, ALIAS=MODEL, USAGE=24, $
    FIELD=BODYTYPE, ALIAS=BODY, USAGE=A9, $
    FIELD=SEATS, ALIAS=STS, USAGE=I3, $
  SEGNAME=EQSEG, SEGTYPE=S, PARENT=MODSEG, $
    FIELD=EQUIP, ECODE, A6, $

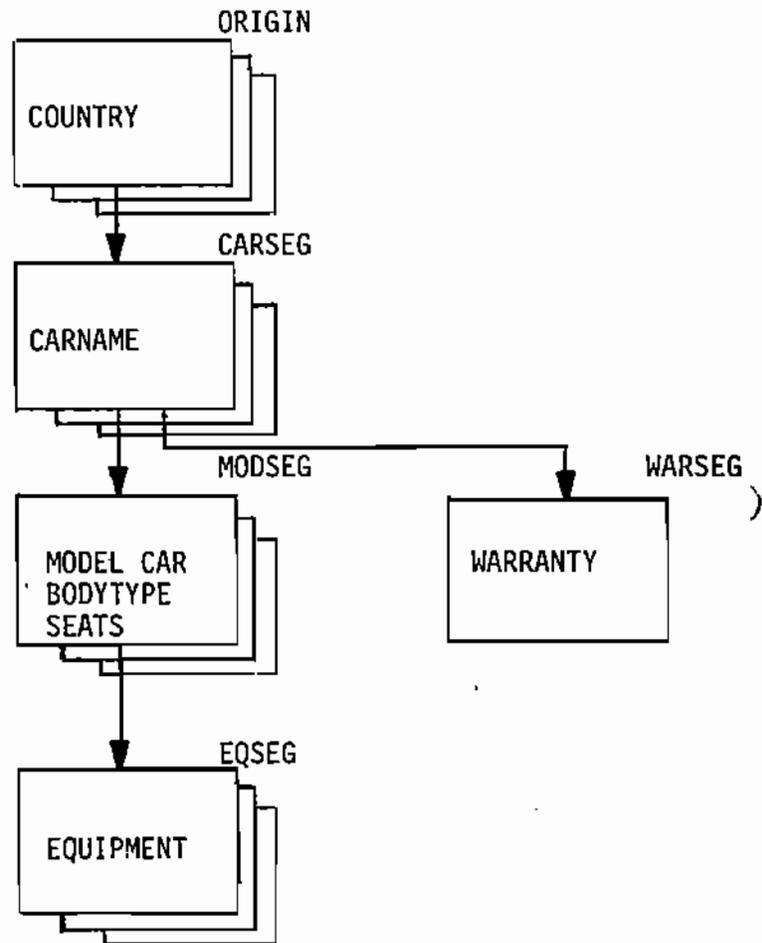
```

MULTI-PATH HIERARCHICAL FILES

- * Segment Parentage
- * Unique Occurrences

Multiple-Path File Structure

An extension to a single path structure, (which has only one top-to-bottom root), is to add alternate top-to-bottom paths. This occurs when a record segment has more than one type of descendent. For instance, suppose warranty information is common to all cars of a given manufacturer. The structure appears as:



This type of structure is described by stating that

SEGNAME=MODSEG, PARENT=CARSEG, SEGTYPE=, \$

Note

SEGNAME=WARSEG, PARENT=CARSEG, SEGTYPE=, \$
FIELD=WARRANTY, ALIAS=WAR, USAGE=A90, \$

Note that both the MODSEG, and the WARSEG specify the same PARENT of CARSEG. In this case there are five segments arranged on four levels of hierarchy. A FOCUS file may have up to 64 different segments arranged on 1 to 16 levels of hierarchy.

Unique Segment

A multi-path structure is sometimes constructed from a collection of data items which are actually all directly related and should be in a single path. This occurs for example, when it is desirable to divide a segment into two or more parts because:

1. Data for some of the items are rarely supplied, hence no storage space for them would be taken if they are in a separate segment.
2. A smaller segment for the 'key' fields means more of them can be fitted onto the same page of storage, hence in cases where a search is needed the retrieval is faster.

This situation where a segment is divided into two or more parts, yet would be treated logically as if all of the items were together, requires that the divided parts be identified. For these segments the SEGTYPE attribute is given the value of 'U', (for Unique segment).

A unique segment is one which can only occur once within its parent. It is 'one to one' with its parent. Any attempt to store two or more instances for the same parent is regarded as an error.

As an example of the usage of a 'U' segment, suppose in a Personnel file the Social Security Number (SSN) is resident on one level, and all information relative to a SSN is placed on a descendent segment; such as FULL NAME, ADDRESS, DEPARTMENT, etc. Another segment with multiple instances of job histories is also a descendent of the SSN.

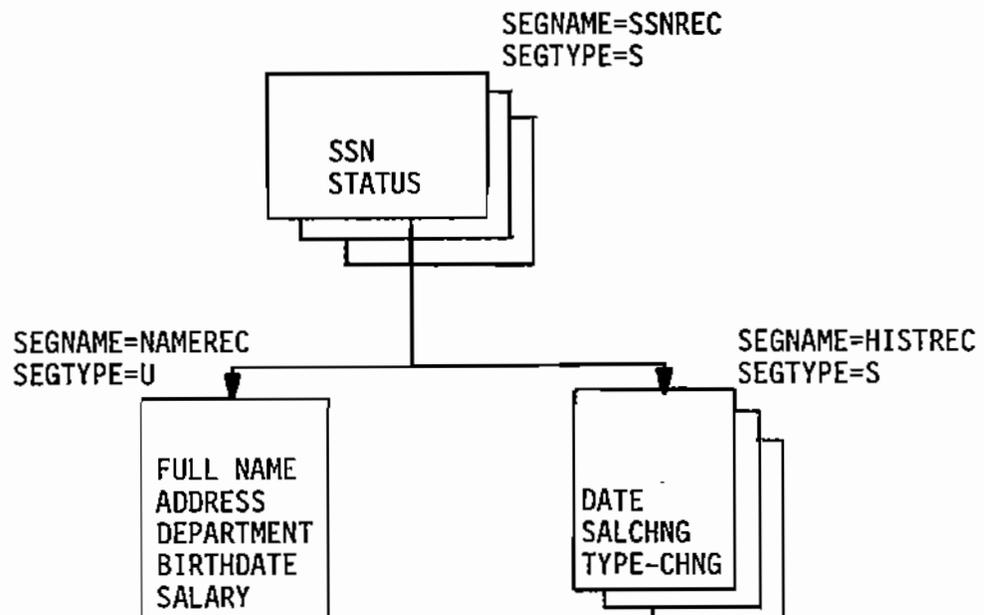


Figure 1: Unique Segment in a Separate Path

Unique Segment (cont'd)

As illustrated in Figure 1, it would clearly be an error to have two FULL NAME's for a given SSN.

The description for the file shown in Figure 1 is:

```
FILENAME=HISTORY, SUFFIX=FOC
  SEGNAME=SSNREC, SEGTYPE=S
    FIELD=SSN, ALIAS=SOCSEG, USAGE=A9, $
    FIELD=STATUS, ALIAS=ST, USAGE=I1, $
  SEGNAME=NAMEREC, SEGTYPE=U, PARENT=SSNREC
    FIELD=FULL NAME, ALIAS=FN, USAGE=S24, $
    FIELD=ADDRESS, ALIAS=ADDR, USAGE=A24, $
    FIELD=DEPARTMENT, ALIAS=DEPT, USAGE=A16, $
    FIELD=BIRTHDATE, ALIAS=DOB, USAGE=I6MDY, $
    FIELD=SALARY, ALIAS=SLY, USAGE=D8.2, $
  SEGNAME=HISTREC, SEGTYPE=S, PARENT=SSNREC
    FIELD=DATE, ALIAS=CDATE, USAGE=I6YMDT, $
    FIELD=SALCHNG, ALIAS=TYPE, USAGE=A2, $
```

Notice in the illustration that the multiply occurring HISTREC was not made a descendent of the NAMEREC. If no data values were ever entered for the items in a given NAMEREC segment, then no storage space in the file would be used.

It is an error to specify descendents of 'U' segments (See Figure 2) because of a special property of SEGTYPE=U segments. When they are linked with other segments on the same hierarchical level as in Figure they will be treated as if they were in a straight line. If they are absent then default values of blanks for alphanumerics and zeros for numbers will be substituted. Hence, SEGTYPE=U provides:

- . saving in storage space for data which may not be present.
- . automatic control over attempts to provide multiple occurrences.
- . default values if no data was ever entered (which is the same as if the fields were part of their parent segment but didn't have data).

Unique Segment (cont'd)

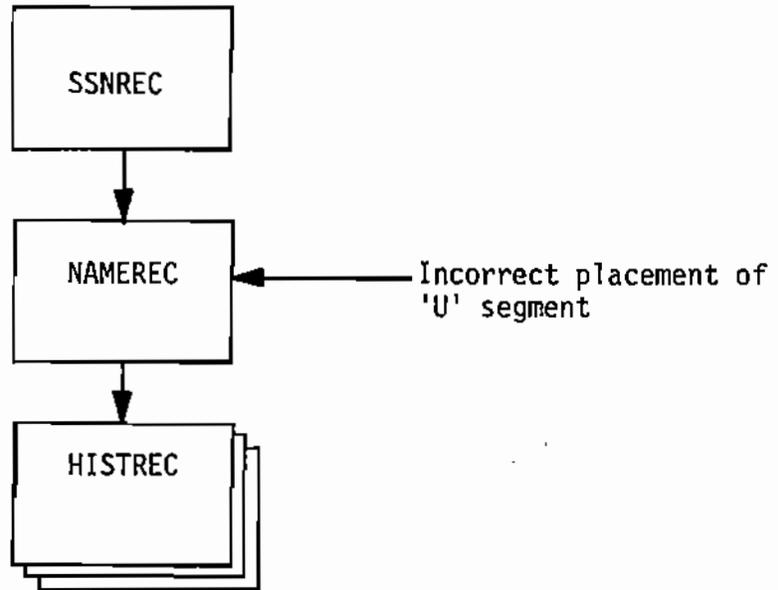


Figure 2: Unique Segment - Between other Segments

UNIQUE CROSS-REFERENCES BETWEEN FILES

- * Shared Segments
- * Segment type 'KU'
- * Cross-Reference key
- * Cross-Reference filename

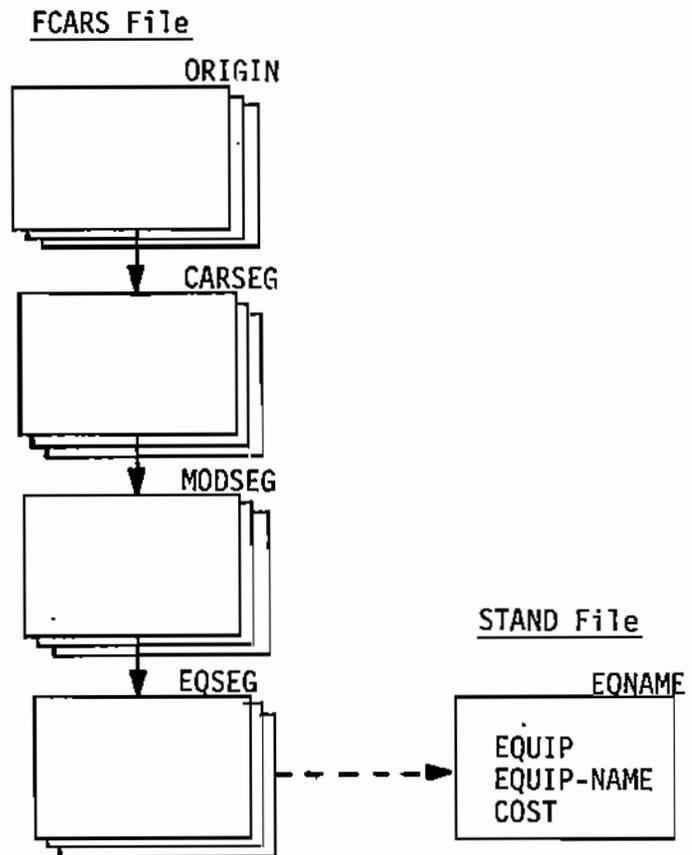
Cross-References Between Files

A data field whose values may be the same in a variety of places is a candidate for access via a cross-reference. In this case the data field and any associated data fields on the same segment are stored only once and this copy of the data is used wherever a reference to it is made.

A typical situation where a cross-reference is useful can be illustrated in the FCARS file used previously. The list of equipment used in each model car is a field named EQUIP on the segment named EQSEG. This field is six characters long (A6) hence it is only a coded abbreviation for the equipment. The full name of the item of equipment, and other related information such as the cost of the equipment is stored in a separate file named STAND. Since the same item of equipment is used in many model cars the ability to cross-reference the data when needed means that the physical segment containing the equipment name, cost, etc, is 'shared'. It is therefore easier to maintain the total list of equipment in this separate file as changes are made only once and are reflected in all places where the data is referenced.

Cross-References Between Files (cont'd)

The file description for the FCARS file would appear with the cross-reference to a segment containing equipment data named EQNAME which is stored in its own file named STAND as:



Cross-References Between Files (cont'd)

In order to communicate that the segment named EQNAME can be cross-referenced by the common data element named EQUIP the segment attributes used for this purpose are:

<u>ATTRIBUTE NAME</u>	<u>ALIAS</u>	<u>LENGTH</u>	<u>MEANING</u>
CRFILENAME	CRFILE	1-8 CHARS	NAME OF FILE CONTAINING DATA
SEGNAME	SEGNAME	1-8 CHARS	NAME OF SEGMENT WITH DATA
CRKEY	VKEY	1-12 CHARS	NAME OF COMMON FIELD ON SEGMENT
SEGTYPE	SEGTYPE	2 CHARS	VALUE IS 'KU', KEYED UNIQUE

The full file description for the FCARS file would be:

```

FILENAME=FCAR, SUFFIX=FOC, $
SEGNAME=ORIGIN, SEGTYPE=S, $
  FIELD=COUNTRY, ALIAS=CTY, USAGE=A20, $
SEGNAME=CARSEG, SEGTYPE=S, PARENT=ORIGIN, $
  FIELD=CARNAME, ALIAS=CAR, USAGE=A14, $
SEGNAME=MODSEG, SEGTYPE=S2, PARENT=CARSEG, $
  FIELD=MODEL, ALIAS=MODEL, USAGE, $
  FIELD=BODYTYPE, ALIAS=BODY, USAGE=A9, $
  FIELD=SEATS, ALIAS=STS, USAGE=I3, $
SEGNAME=EQSEG, SEGTYPE=S, PARENT=MODSEG, $
  FIELD=EQUIP, ECODE, A6, $
SEGNAME=EQNAME, PARENT=EQSEG, SEGTYPE=KU
  CRFILE=STAND, CRKEY=EQUIP, $ ←————— Note

```

Note that only the name of the cross-referenced segment is provided; the list of fields on the segment is already known from the cross-reference file description, from the STAND file description in this example.

Cross-References Between Files (cont'd)

Since the two files share a common segment, the description of the cross-reference file must be available whenever the referencing file is used. (There need not be any data in the referenced file).

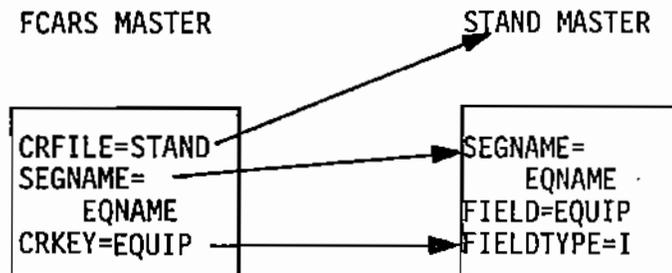


Figure 3: File Description With Shared Segment

Preparing a Segment for SharingFIELDTYPE=I

In order for a segment to be cross-referenced one or more fields must have a FIELDTYPE=I. This means that an index is to be constructed for the field values. It is through the use of these indexes that other files may locate and use a segment. Any number of fields may be indexed on a segment, although only those which have common values in other files are practical candidates. For example, in the equipment data file called STAND the fields on the first segment are:

SEGNAME=EQNAME

EQUIP (I)
EQUIPNAME
COST

The FCARS file can locate a match to a value for its field named EQUIP by consulting the index in the STAND file for the field also named EQUIP. When the type of cross-reference is 'KU' (keyed unique) it means that only one matching value will be found and one corresponding segment retrieved. In actuality, the first time the shared segment is retrieved its location is remembered by the cross-referencing file, and thereafter the index is consulted only if a change in value occurs. The index maintenance is entirely automatic and invisible and can be thought of as a list of pointers to data values stored apart from the data but maintained with it.

The presence of the index is crucial for the operation of the cross-referencing facilities. Any number of external sources may locate and thereby share a segment because of it.

Preparing a Segment for Sharing FIELDTYPE=I (cont'd)

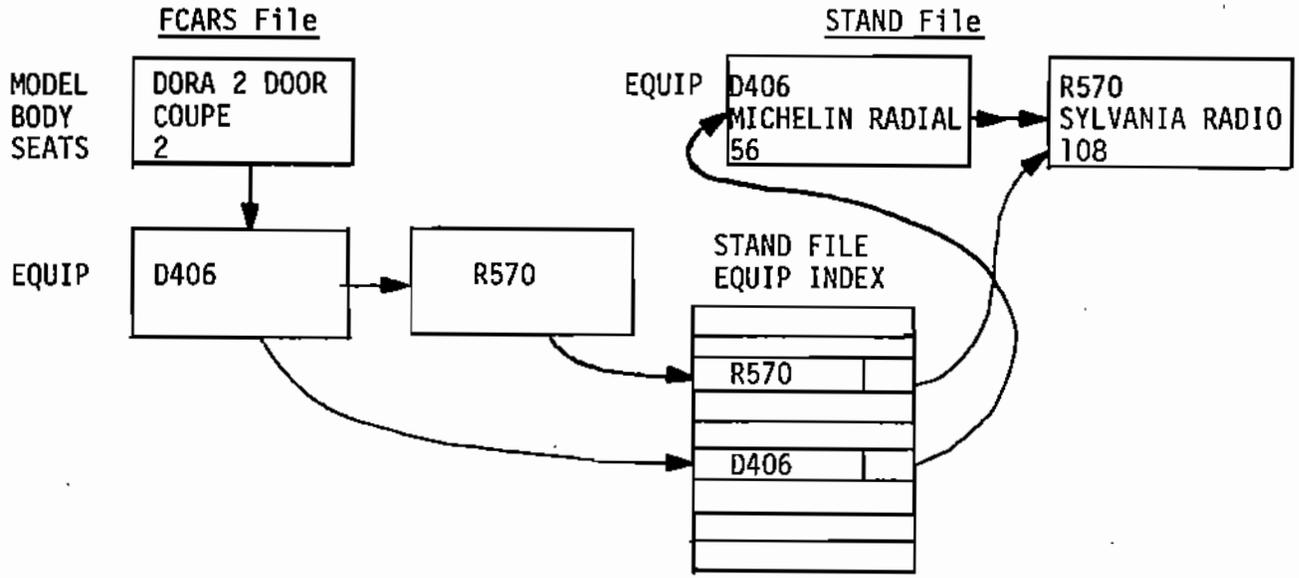


Figure 4: Locating a Matching Value Through an Index

New files which have data items in common with indexed fields in existing files can be added at any time.

Decoding:

A common problem solved by the use of a unique cross-reference is the simple decoding of a 'coded' value into its 'decoded' value. For instance, a 'district number' into its equivalent 'district name'. The shared segment in this case has two fields, 'district number' which is indexed, and 'district name'. The procedure is applicable for 10 values of 'district number' or 10,000 values. However, for small lists, under 20 items, the DEFINE mode function named DECODE may be more appropriate. (See Report Preparation - Defining Temporary Variables).

CROSS-REFERENCING FROM ONE FILE TO
ALL OCCURRENCES IN ANOTHER FILE

* Segment type 'KM'

Cross-Referencing Multiple Instances

Since an index can be constructed containing information of all occurrences of each value of a field it is possible to cross-reference a segment based on a field value, and retrieve all occurrences of it. The file description attribute used for this purpose is:

ATTRIBUTE NAME	ALIAS	LENGTH	MEANING
SEGTYPE	SEGTYPE	2 CHARS	VALUE IS 'KM', keyed multiple

An illustration of the use of this facility would occur if in the STAND file example, the question is asked, "which car models use a given item of equipment". That is, starting from the field EQUIP, in the STAND file find all like values of EQUIP in the FCARS file. This requires the indexing of the field named EQUIP in the FCARS file so that it can be cross-referenced from the STAND file. The file structure would appear as:

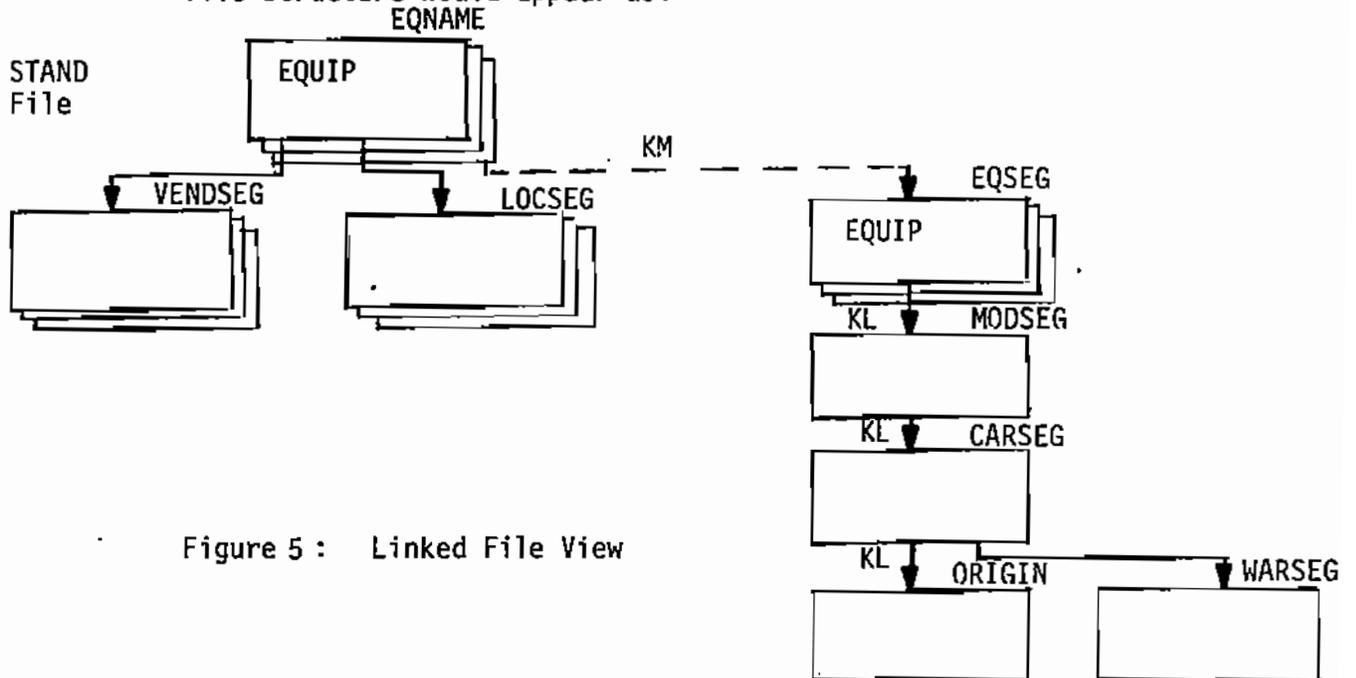
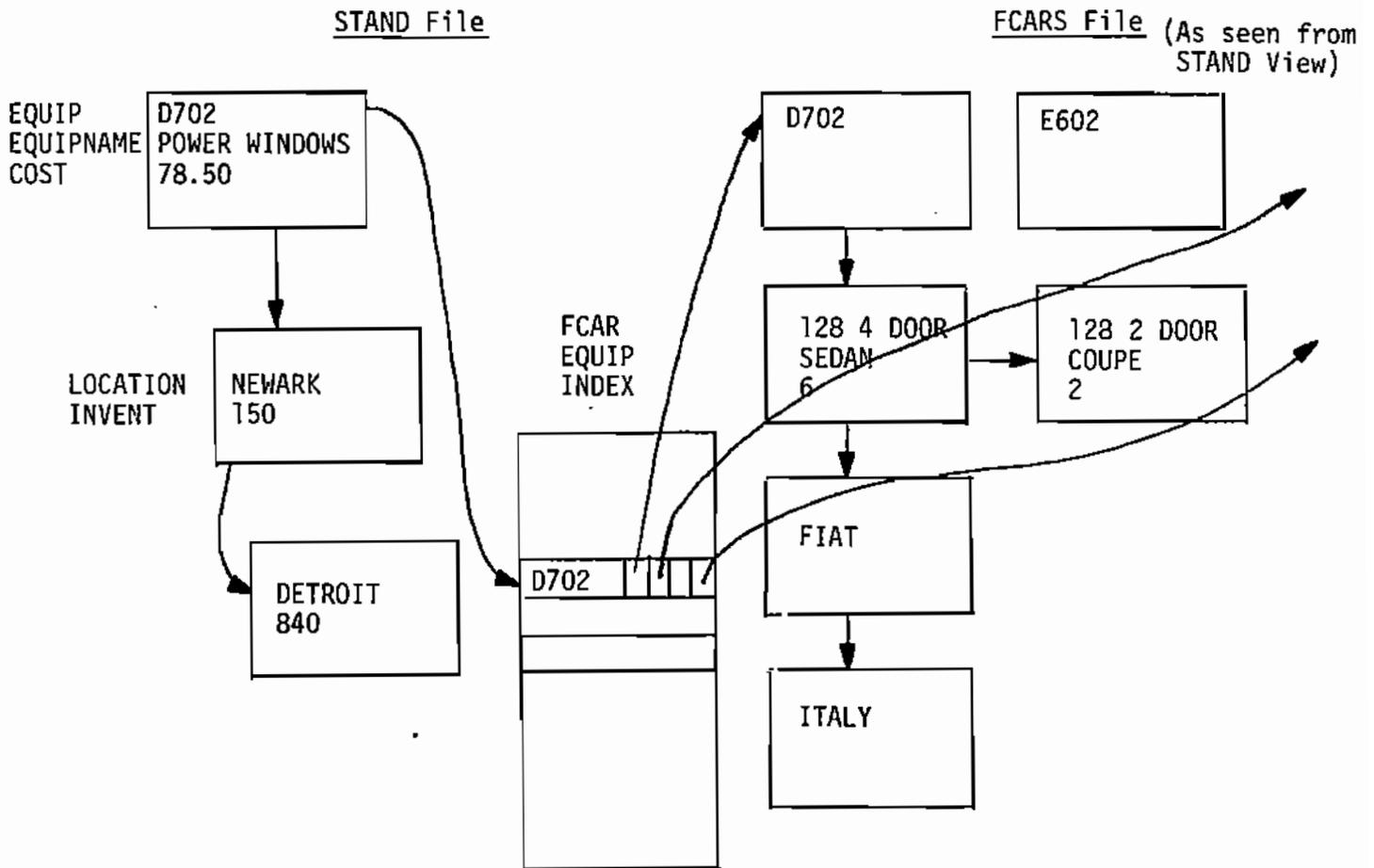


Figure 5 : Linked File View

One practical advantage of this type of view is the rapid access it affords to a small number of selected occurrences among a large number. For instance if there are 200 different items of equipment and 1000 car models with an average of 30 items per model then it would be faster to start with the STAND file, find the particular item desired from the 200, and through it find directly the approximately $30 \times 1000 / 200 = 150$ car models having this item of equipment, rather than searching the chain of 30000 equipment-model combinations for the 150.

A picture of the way the data is connected in the example is shown below.



The first time a request to the STAND file encounters a value which has not been cross-referenced before the appropriate index is consulted and the location in the index where the list of matching values is found is stored back in the STAND file. Thereafter, all requests for that cross-reference value goes directly to the list. The maintenance of the changes to the index and its usage is entirely automatic and invisible.

The complete file description for the STAND file shown in Figure 4 with the cross-references to the bottom of the FCARS file is:

```
FILENAME=STAND, SUFFIX=FOC, $
SEGNAME=EQNAME, SEGTYPE=S, $
  FIELD=EQUIP, ALIAS=EQUIPCODE, FORMAT=A6, FIELDTYPE=I, $
  FIELD=EQUIPMENT, ALIAS=EQUIPNAME, FORMAT=A40, FIELDTYPE= , $
  FIELD=FCOST, ALIAS=COST, FORMAT=D8.2, $
SEGNAME=VENDSEG, SEGTYPE=S, PARENT=EQNAME, $
  FIELD=SUPPLIER, ALIAS=VENDNAME, FORMAT=A24, $
  FIELD=VLOCATION, ALIAS=LOC, FORMAT=A24, $
SEGNAME=LOCSEG, SEGTYPE=S, PARENT=EQNAME, $
  FIELD=WAREHOUSE, ALIAS=STOCKPT, FORMAT=A20, $
  FIELD=INVENT, ALIAS=STOCK, FORMAT=I6, $
SEGNAME=EQSEG, PARENT=EQNAME, SEGTYPE=KM
  CRKEY=EQUIP, CRFILE=FCARS, $
SEGNAME=MODSEG, PARENT=EQSEG, SEGTYPE=KL,
  CRFILE=FCARS, $
SEGNAME=CARSEG, PARENT=MODSEG, SEGTYPE=KL,
  CRFILE=FCARS, $
SEGNAME=ORIGIN, PARENT=CARSEG, SEGTYPE=KL,
  CRFILE=FCARS, $
SEGNAME=WARSEG, PARENT=CARSEG, SEGTYPE=KL
  CRFILE=FCARS, $
```

The segment attribute SEGTYPE=KL means a segment reached through a key then through a linkage with the keyed segment. See the next section, 'Linking to Segments Accessible From a Cross-Referenced Segment' for additional information.

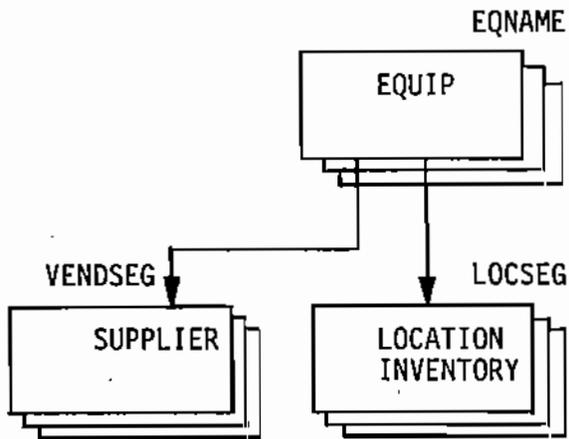
LINKING TO SEGMENTS ACCESSIBLE FROM A CROSS-REFERENCED SEGMENT

* Segment type of 'KL' and 'KLU'

Sharing Linked Segments

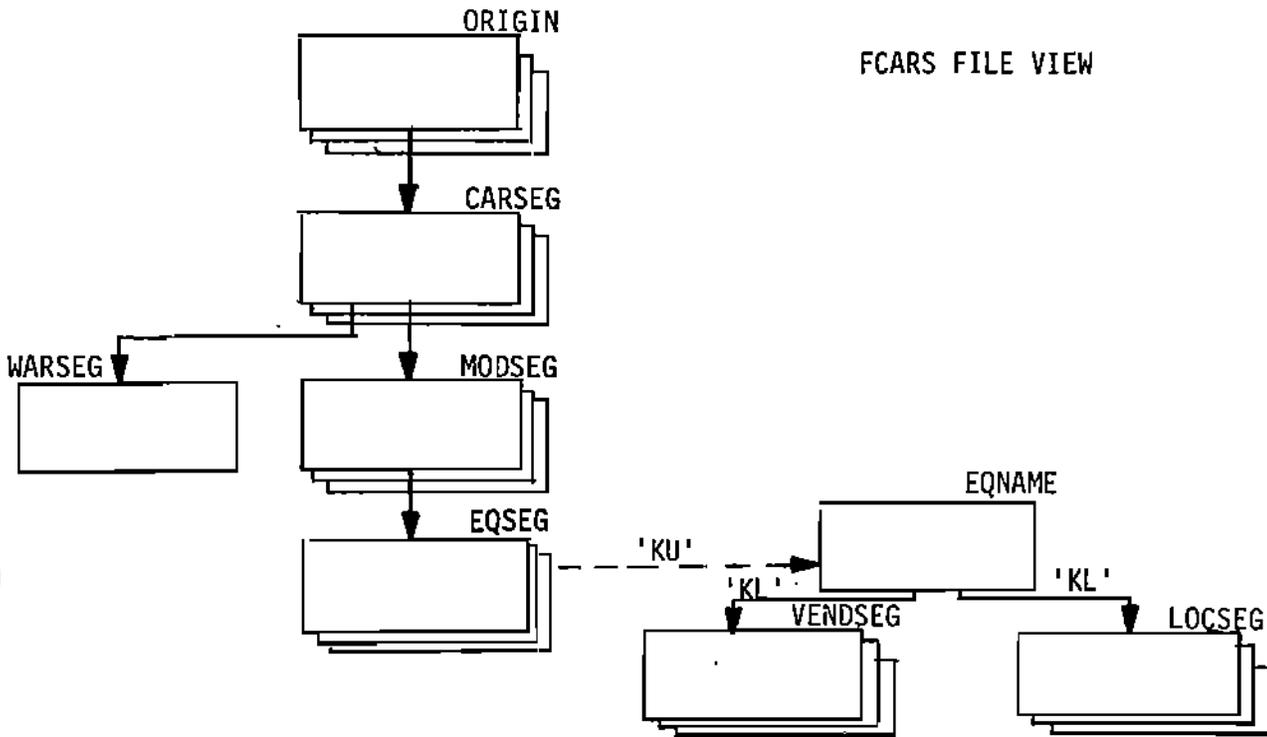
A file which cross-references a segment from another file may then retrieve any segment which is related (linked) to the shared segment. This includes all descendent segments, as well as the parents of the cross-referenced segment.

An illustration of the use of this facility would occur if in the STAND file example the inventory of the equipment at various warehouse locations was maintained, plus a list of suppliers for the equipment (on a parallel level). The file structure for the STAND file would be:



Sharing Linked Segments (cont'd)

From the point of view of the FCARS file the entire collection of information would appear as:



A report request from the FCARS file has available all of the information in both the FCARS and STAND files. All of the field names in both files are available at report request time as if the data was all in one physical file.

When the FCARS description is created it cross-references the segment named EQNAME, then it refers to any segment linked to the cross-referenced segment as SEGTYPE 'KL' (keyed through linkage). The field names for linked segments do not have to be provided in the FCARS description as they are already described in the STAND file.

Sharing Linked Segments (cont'd)

The full file description for FCARS is:

```

FILENAME=FCAR, SUFFIX=FOC, $
SEGNAME=ORIGIN, SEGTYPE=S, $
  FIELD=COUNTRY, ALIAS=CTY, USAGE=A20, $
SEGNAME=CARSEG, SEGTYPE=S, PARENT=ORIGIN, $
  FIELD=CARNAME, ALIAS=CAR, USAGE=A14, $
SEGNAME=MODSEG, SEGTYPE=S2, PARENT=CARSEG, $
  FIELD=MODEL, ALIAS=MODEL, USAGE, $
  FIELD=BODYTYPE, ALIAS=BODY, USAGE=A9, $
  FIELD=SEATS, ALIAS=STS, USAGE=I3, $
SEGNAME=EQSEG, SEGTYPE=S, PARENT=MODSEG, $
  FIELD=EQUIP, ECODE, $
SEGNAME=EQNAME, PARENT=EQSEG, SEGTYPE=KU
  CRFILE=STAND, CRKEY=EQUIP, $
SEGNAME=LOCSEG, PARENT=EQNAME, SEGTYPE=KL
  CRFILE=STAND, $
SEGNAME=VENDSEG, PARENT=EQNAME SEGTYPE=KL
  CRFILE=STAND, $

```

The file description attribute used to access a 'linked' segment is:

<u>ATTRIBUTE</u> <u>NAME</u>	<u>ALIAS</u>	<u>LENGTH</u>	<u>MEANING</u>
SEGTYPE	SEGTYPE	2-3 CHARS	Value is 'KL' or 'KLU'

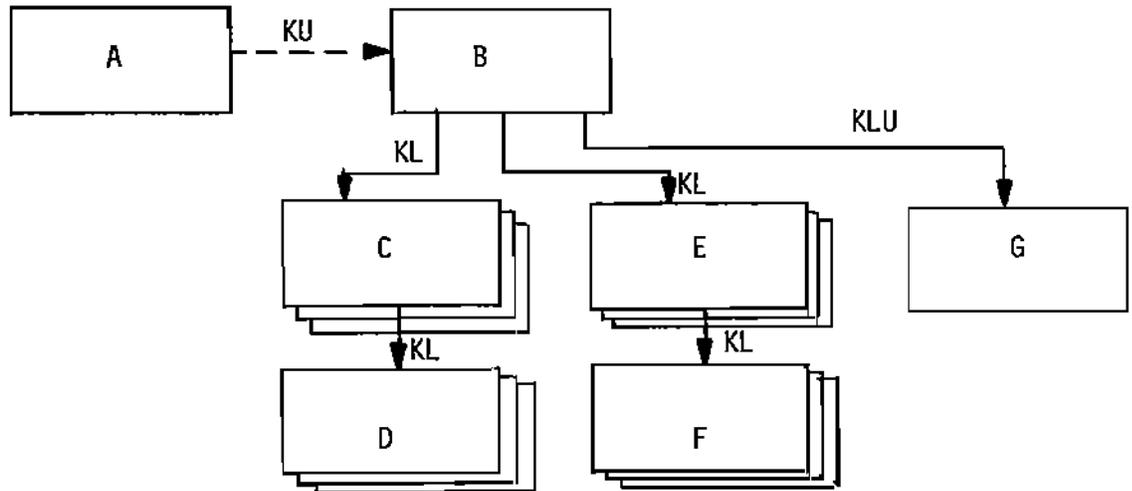
Segment type KLU:

The segment type of KLU differs from KL only in the information contained in the significance of the letter 'U' (unique). This indicates that a single instance will be retrieved and all of the fields can be considered to be in a one-to-one relation to its parent. This is analogous to the 'U' segments. (See SEGTYPE='U'). For instance if the STAND file had a 'U' type of segment as a descendent of EQNAME then when accessed via its linkage through EQNAME it would be labeled 'KLU' rather than 'KL'.

Sharing Linked Segments (cont'd)

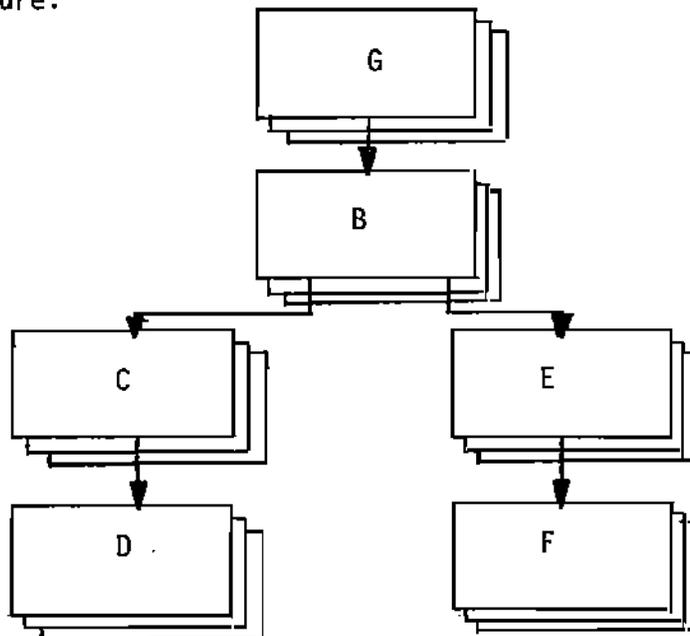
Hierarchies of Linked Segments:

A segment may be accessed through its linkage with another segment. Hence, a 'KL' segment may lead to other 'KL' segments. Illustratively this appears as:



The letters on the arrows are the SEGTYPE.

In the physical file which has been cross-referenced it is possible that segment 'G' is the parent of 'B'. This is the real related segment structure:



Sharing Linked Segments (cont'd)

When a file view cross-references a segment and the parent of the segment is accessed it is described like all linked segments as a descendent of the cross-referenced segment. This is entirely consistent with the way we 'view' the data, which in this case is from the cross-referencing file's point of view. In the next section it will be illustrated how this helps us 'invert' a file and view it bottom up, rather than top down.

SPECIAL TOPICS

- * Re-naming cross-referenced fields
- * Cross-referencing the same segment in multiple places
- * Complex File Structures
- * The USE Command
- * Combining Partitioned Files
- * CMS Extension files
- * Data Administrator Package (DAP)

Re-naming Cross-Referenced Fields

The cross-referencing facility only requires that the segment name in the cross-referenced file be provided. The names of the data fields are then taken from the cross-referenced file. It is possible, however, to explicitly provide a description of the fields on the cross-referenced segment. This allows new names to be selected, or some fields dropped from the description (but the space accounted for with a filler field of equal length to the missing fields). Most importantly, it allows the same cross-referenced segment to be used in several places in one file. The fields on the segment can then be given different names in each position so that they can be differentiated in report requests.

An example of this usage occurs in the situation shown below. Two fields in the structure refer to the names of people. The full information about people are stored in another file. Hence, the same segment is cross-referenced from two places.

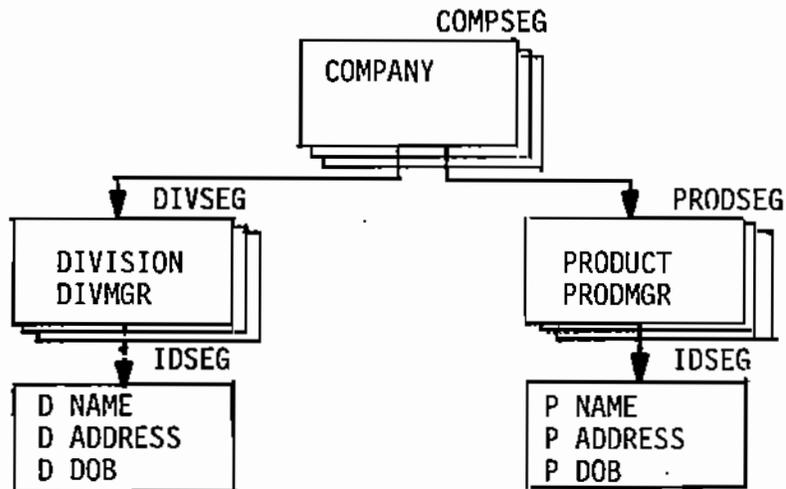


Figure 6 : Same Segment Cross-Referenced from More Than One Place

Re-naming Cross-Referenced Fields (cont'd)

The following conventions must be used:

- The ALIAS attribute of the common field must be the same in all places and must be the same as the FIELDNAME used in the cross-referenced file. For instance:

```
FIELD=DIVMGR, ALIAS=NAME, ...
```

```
FIELD=PRODMGR, ALIAS=NAME, ...
```

- The field attributes for the cross-referenced segment are typed after the name of the segment in the order they actually occur in the cross-referenced segment. There must not be a terminator (, \$) typed after the segment attributes and before the field attributes. i.e.

```
SEGNAME=IDSEG, SEGTYPE=KU, CRKEY=
  CRFILE=PFILE
  FIELD=DNAME, ALIAS=NAME, FORMAT=A12, FIELDTYPE=I, $
```

- The common field on the cross-referenced segment must have a FIELDTYPE=I, and the same ALIAS as the FIELDNAME in the real segment.

The description of the file used as an example is:

```
FILENAME=COMPDAT, SUFFIX=FOC, $
```

```
SEGNAME=COMPSEG, SEGTYPE=S, $
```

```
  FIELD=COMPANY, ALIAS=CPY, FORMAT=A40, $
```

```
SEGNAME=DIVSEG, PARENT=COMPSEG, SEGTYPE=S, $
```

```
  FIELD=DIVISION, ALIAS=DV, FORMAT=A20, $
```

```
  FIELD=DIVMGR, ALIAS=NAME, FORMAT=A12, $
```

```
SEGNAME=IDSEG, PARENT=DIVSEG, SEGTYPE=KU
```

```
  CRKEY=DIVMGR, CRFILE=PFILE ←—————Note
```

```
  FIELD=DNAME, ALIAS=NAME, FORMAT=A12, FIELDTY=I, $
```

```
  FIELD=DADDRESS, ALIAS=DAS, FORMAT=A24, FIELDTY=, $
```

```
  FIELD=DBOB, ALIAS=DOB, FORMAT=I6MDYT, $
```

```
SEGNAME=PRODSEG, PARENT=COMPSEG, SEGTYPE=S, $
```

```
  FIELD=PRODUCT, ALIAS=PDT, FORMAT=A8, $
```

```
  FIELD=PRODMGR, ALIAS=NAME, FORMAT=A12, $
```

```
SEGNAME=IDSEG, PARENT=PRODSEG, SEGTYPE=KU
```

```
  CRKEY=PRODMGR, CRFILE=PFILE ←—————Note
```

```
  FIELD=PNAME, ALIAS=NAME, FORMAT=A12, FIELDTY=I, $
```

```
  FIELD=PADDRESS, ALIAS=PAS, FORMAT=A20, FIELDTY=, $
```

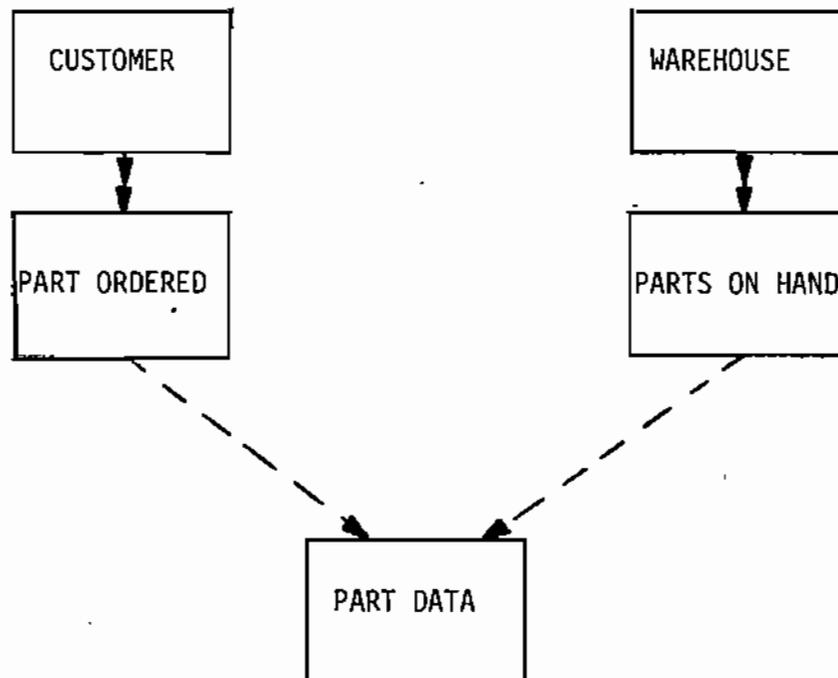
```
  FIELD=PDOB, ALIAS=DOB, FORMAT=I6MDYT, $
```

Complex File Structures

The combination of hierarchical relations and cross-references provides a wide variety of options in structuring a given data base. The following two examples illustrate two less obvious situations where these tools can be creatively used.

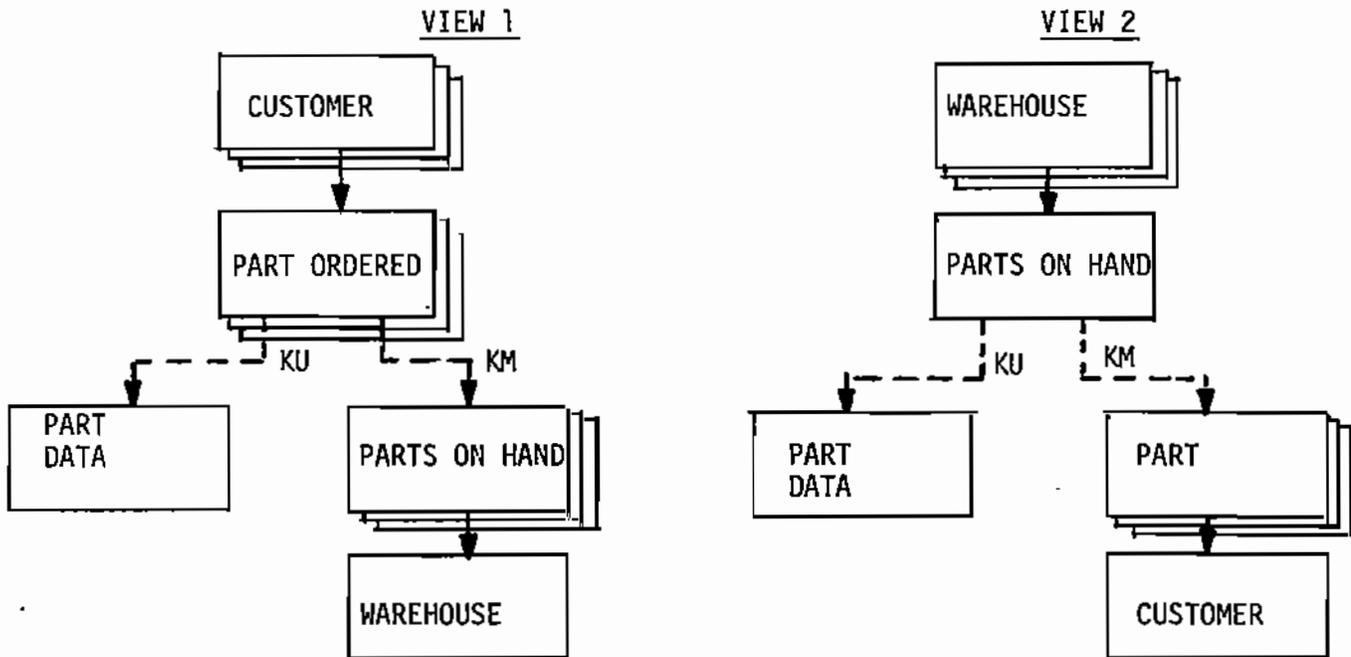
Multiple Parents:

The choice of a hierarchical structure to arrange the segments in the following application requires that one segment have two parents. The application keeps track of customer orders for parts, and warehouse inventory of parts, and general part information. Schematically this appears as:



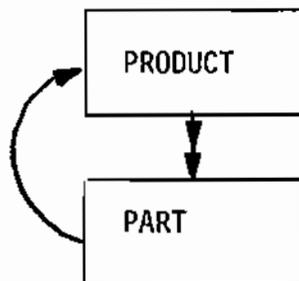
Multiple Parents: (cont'd)

There are several FOCUS descriptions appropriate for this application; one such design is:



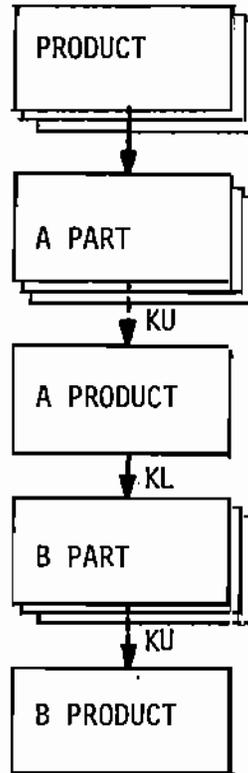
Recursive Reuse of a Segment:

In rare cases a file may cross-reference itself. For instance, a file of products each having a list of parts which compose the product, where a part may itself be a product and have sub-parts. This may continue to several levels of sub-parts. Schematically, this would appear as:



Recursive Reuse of a Segment:

A FOCUS description for this case, shown for two levels of sub-parts is:



A request statement of the following type could be issued:

```
LIST APART AND APRODUCT AND BPART AND BPRODUCT  
IF PRODUCT IS RD104
```

Other request statements could be formulated to 'explode' end quantities needed, etc.

Combining FOCUS Files at Run TimeThe USE Command

The FOCUS 'USE' command is used in three situations.

1. To identify a data file whose CMS 'filemode' is not 'A1'. That is, a file which is not on the default 'A' disk.
2. To identify a data file whose CMS 'filetype' is not FOCUS.
3. To concatenate separate data files all having the same FOCUS filename into one logical file for report preparation purposes.

The syntax is:

```
USE
filename filetype filemode
filename filetype filemode
:
:
END
```

Example: Data File on 'C' Disk

```
USE
FCARS FOCUS C1
END
```



Example: Data File Not Filetype 'FOCUS'

```
USE
FCARS TEST B1
END
```

The USE Command (cont'd)Example: Several Files

```
USE
FCARS FOCUS A1
STAND FOCUS B1
DEALER NAMES C1
END
```

Partitioned Data Files:

A partitioned data file is one which occurs in separate parts. Each part is a separate file having the identical file description. Each part is managed along, and perhaps simultaneously with other parts if they are on separate disks, yet for report preparation purposes can be logically combined.

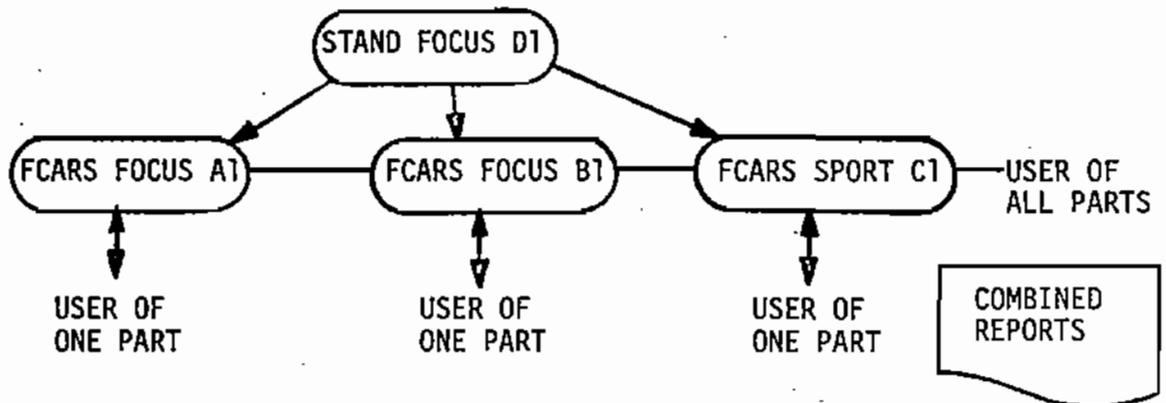
The advantages of this type of design is that it allows each user of the data base access to only one physical part. This part is smaller, hence activities with it will usually be faster. Any errors in data entry to one part will not inconvenience other users if a backup edition of one part has to be restored. When historical copies of a file are kept because new data starts a new file, i.e. current year, last year files, then it is easy to produce combined reports without having to keep the historical data always available.

The different partitions of the file have the same filename and are identified in the USE command by their full CMS fileid 'filename file-type filemode'. For instance:

```

USE
FCARS FOCUS A1
FCARS FOCUS B1
FCARS SPORT C1
STAND FOCUS D1
END
    
```

A picture of the environment described by this example is:



Extended Size Files

On most CMS systems the maximum size a single physical file can attain is 12.8 million characters. FOCUS files may attain a maximum size of 128 million characters on CMS. In order to remove this CMS restriction a single FOCUS file automatically stops growing at 12 million characters, and an extension to the file is started under the same filename and filemode, but with the suffix 'X1' attached to the filetype. When the 'X1' extension reaches its physical limit an 'X2' suffix file is started, and so on up to 'X9'. Hence, a FOCUS file with 30 million characters will be found on the disk in three physical CMS files. For instance:

```
FCARS FOCUS
FCARS FOCUSX1
FCARS FOCUSX2
```

The user should ignore the extension files when issuing a USE command. Only the original file identity is needed in USE commands.

When CMS utility commands are issued remember to include the extension files. For instance, when a backup copy of a data base is made with the CMS TAPE DUMP utility.

Data Base Administrator Tools

The Data Administrator Package (DAP):

The material in this section is included for planning purposes only. The Data Administrator Package (DAP) is not part of FOCUS release 1.

Data files which are accessed by multiple users either for the purpose of inquiring or updating may require controls on the access environment. DAP provides the facility to name each user class, and state the prohibited actions and restrictions which apply for each class.

The access environments are:

- READ
- UPDATE
- INPUT/DELETE

The restrictions for each of these environments are:

- File access
- Field access
- Field Select Value
- Validate test value
- Shared Usage

The data base administrator constructs a decision table showing the restrictions applied to each user class in each access environment. This table can be thought of as a matrix. A sample is shown below:

CONTROL	User Class					
	TED			BELL		
	READ	UPDATE	IN/DEC	READ	UPDATE	IN/DEL
FILE	Y	N	N	Y	Y	N
FIELD						
SALARY	N			Y	N	
PROMOTE	N			N		
SELECT						
DIVTEST	Y			N		
VALIDATE						
GOODATA					Y	

Data Administrator Package (DAP): (cont'd)

The select conditions are screening phrases applied to every request made. For instance:

```
DIVTEST=DIVISION EQ 'EAST'
```

This limits user class 'TED' for instance to data from one division. The validate conditions are automatic application of data validity tests to any new transactions.

One of the user classes is reserved to the data base administrator who may change the control table.

The controls on the data 'travel' with the data, hence cannot be bypassed, so the administrator may allow access to the data to users who may want to cross-reference it from their own systems. There are restriction attributes for a shared usage environment.

The DAP sub-system is designed for expansion in the types of control attributes and is meant to provide a very comprehensive facility for use by a data base administrator.

FILE DESCRIPTION ATTRIBUTES

- * FILENAME
- * SUFFIX
 - * SEGNAME
 - * PARENT
 - * SEGTYPE
 - * CRKEY
 - * CRFILENAME
 - * FIELDNAME
 - * ALIAS
 - * USAGE FORMAT
 - * FIELDTYPE

Attribute: FILENAME
Alias: FILE
Length: 1-8 CHARS
Permitted Values: All characters and digits but no embedded blanks.
Example: FILENAME=EQUIP

Function: The FILENAME is used to identify a file description, and later the data entered under that file description. The description is stored in a CMS file of the same name and with filetype of MASTER. Hence, for example, the data file called FCARS has its file description stored in FCARS MASTER. (The top line of this description file should also have FILENAME=FCARS). The first time the command MODIFY is issued a data file with the selected FILENAME and with filetype of FOCUS will be created. The following message is displayed on the terminal for a new file.

```
NEW FILE 'filename FOCUS A1' ON nn/nn/nn AT nn:nn:nn
```

The data file may be transferred to another disk and given another filetype once it has been created. (Even if no records were actually entered). However, if the default filetype of 'FOCUS' is changed, or the filemode is not 'A1', then the FOCUS USE command is needed to identify the location of the data file. See the section 'The USE Command' for details.

Changes: If no segments in the file are cross-referenced by other files then the FILENAME may be changed, and both the MASTER file and data files renamed. If a segment within the file is cross-referenced then all other file descriptions which mention this file must also be changed to refer to the new filename.

Attribute: SUFFIX
Alias: FILESUFFIX
Length: 1-4 CHARS
Permitted Values: FOC, COM, FIX, ⌀
Example: SUFFIX=FOC

Function: The FOCUS report writer can be used on non-FOCUS files. In order to identify the type of file the description applies to the SUFFIX is given one of four values.

Value

FOC	FOCUS file
COM	Comma delimited external file
FIX	Fixed format external file
⌀	Fixed format external file

Note: FOCUS files must have SUFFIX=FOC.

Changes: No changes can be made to this attribute.

Attribute: SEGNAME

Alias: SEGMENT

Length: 1-8 CHARS

Permitted Values: All characters and digits

Example: SEGNAME=MODSEG

Function: The SEGNAME identifies a collection of data fields. Segments which have a relationship to each other must have unique names within a given file description. Segments which are cross-referenced in a file description may have the same names as either other cross-referenced segments, or related segments.

Changes: If no other file description cross-references the segment then its name may be changed and all descendent segment which refer to it as a PARENT must also be changed. If any cross-reference exists to the segment its name cannot be changed.

Attribute: PARENT
Alias: PARENT
Length: 1-8 CHARS
Permitted Values: All characters and digits.
Example: PARENT=CARSEG

Function: The PARENT is the name of the segment which is the hierarchical parent or 'owner' related to the current segment.

In the file description the information describing the parent must precede any reference to it as a parent.

The first segment, or 'root' segment in a file cannot have a parent by definition, but the name 'SYSTEM' may be used, or the attribute left blank. Every other segment must have a parent named.

Changes: The parentage of a segment may not be re-assigned.

Attribute: SEGTYPE
 Alias: SEGTYPE
 Length: 1-4 CHARS
 Permitted Values: Ø, U, S, SH, Si or SHi i=1, 255, KU KM KL KLU
 Example: SEGTYPE=S2

Function: The SEGTYPE identifies some basic characteristics of related segments, or cross-referenced segments.

Related segments may have the values of:

<u>Value</u>	<u>Meaning</u>
Ø	Segments of data are stored on a first come first location basis.
S	Segments of data logically sequenced in a low-to-high order using the first field on the segment as the sequence key.
Si	i=1, 255 Segments of data logically sequenced in a low-to-high order using the first 'i' fields on the segment as sequence keys. For instance, S3.
SH	Segments of data logically sequenced in a high-to-low order using the first data field on the segment as a sequence key.
SHi	i=1, 255 Segments of data logically sequenced in a high-to-low order using the first 'i' fields on the segment as the sequence keys. For instance, SH2.
U	A single instance of the segment will be allowed under its particular parent instance. It is 'unique' to its parents.
S0	S (zero) There is no attempt to match data values. Segments are included one after the other in all cases. Hence, segments are not maintainable. Used in cases where a file is being built once for inquiry only as speed of file construction is main advantage. Also used for segments consisting solely of text.

SEGTYPE (cont'd)

Cross-referenced segments may have values of:

<u>Value</u>	<u>Meaning</u>
KU	The common key field value matches a unique instance in the cross-referenced data file.
KM	The common key field value matches multiple instances in the cross-reference data file.
KL	The segment named is reached through a linkage with its named parent. The parent, however, is reached via a KU, KM, or other KL reference.
KLU	Same as KL except only a 'unique' instance in cross-referenced file has SEGTYPE=U, or if the parent named in this file is actually a child of the segment in the cross-referenced file.

Changes: Segtypes of S or Sh may be made blank.

Attribute: CRKEY
 Alias: VKEY
 Length: 1-12 CHARS
 Permitted Values: All character and digits.
 Example: CRKEY=EQUIPMENT

Function: The attribute CRKEY (cross-reference key) is used only when the segment described is not physically present in the file but is retrieved as a cross-reference. When the SEGTYPE is either KU or KM, then CRKEY is given the name of the data field in a parent segment (or grand parent) whose value is to be used as the common lookup key. The full fieldname or alias name may be used.

Example: Use of CRKEY

FILENAME=FCARS, SUFFIX=FOC

SEGNAME=CARSEG, SEGTYPE=S

FIELD=MODEL-CAR, ALIAS=MODEL, USAGE=A20, \$

FIELD=BODY-TYPE, ALIAS=BODY, USAGE=A8, \$

FIELD=SEATS, ALIAS=SE, USAGE=I4, \$

FIELD=DEALER COST, ALIAS=DCOST, USAGE=D7.0, \$

SEGNAME=INVSEG, PARENT=CARSEG, SEGTYPE=KM

CRFILE=INVENT, CRKEY=MODEL-CAR, \$

Note

The cross-referenced file must contain the requested segment, and have a fieldname with the same name as the CRKEY. For instance,

FILENAME=INVENT, SUFFIX=FOC

SEGNAME=INVSEG, SEGTYPE=S

FIELD=LOCATION, ALIAS=LOC, USAGE=A15, \$

FIELD=MODEL-CAR, ALIAS=MODEL, USAGE=A20, FIELDTYPE=I, \$

Changes: As long as the SEGTYPE remains KU, or KM the CRKEY may be changed.

Attribute: CRFILENAME

Alias: CRFILE

Length: 1-8 CHARS

Permitted Values: All characters and digits.

Example: CRFILE=STAND

Function: The CRFILE is the name of the file in which the cross-referenced segment is described.

Cross-referenced files do not have to have any data, but their description must be present.

Changes: The name may be changed at any time.

Attribute: FIELDNAME

Alias: FIELD

Length: 1-12 CHARS

Permitted Value: All characters and digits.

Example: FIELD=INVENTORY

Function: The FIELDNAME identifies the data items in a file. Names must be unique within a file.

The full fieldname is used as the default title for the data printed on reports. Hence, names representative of the data should be selected.

Changes: If there is no index constructed for the field, that is FIELDTYPE is not 'I', then the name may be changed.

Attribute: ALIAS

Alias: SYNONYM

Length: 1-12 CHARS

Permitted Values: All characters and digits but no special characters.

Example: ALIAS=CTY

Function: The ALIAS is an alternate name to identify a data item. Since references to data items are via their names, or aliases, or shortest unique truncation it is useful to make the ALIAS a short abbreviation representative of the data. Short easy to type names are recommended, with no embedded blank spaces or special characters of +, -, \$, *, /, ', =.

i.e. PART CODE or PART-CODE should be PARTCODE or better yet PC.

Changes: Aliases can be changed at any time.

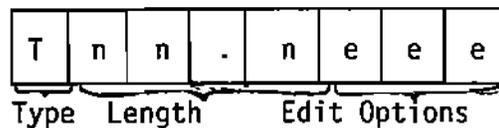
Attribute: USAGE FORMAT

Alias: FORMAT

Length: 1-8 CHARS

Function: The USAGE FORMAT is used to describe the type and length of stored data values and the edit options to be used when the data is printed. Every field in a file must have USAGE information.

3 Parts of a Format



Numerical data may have one of four types of values. In general computational values should use type D as this may be specified both with and without decimals places. Alphanumeric data has a type 'A'. Fields which are dates are handled by a separate set of options.

I Integer $\leq \text{length} \leq 9$

The data is composed of the digits, 0 to 9. The values are stored internally as a full word binary integer.

Example:

I6 e.g. 4316, -617 etc.

Display options may be used to edit the numbers in various ways.

Example:

<u>Option</u>	<u>Format</u>	<u>Data</u>	<u>Display</u>
Comma inclusion	I6C	41376	41,376
Zero suppression	I6S	0	
Bracket negatives	I6B	-64187	(64187)
Credit negative	I8R	-3167	3167 CR
Leading zeros	I4L	31	0031

USAGE FORMAT (cont'd)

Example:

<u>Option</u>	<u>Format</u>	<u>Data</u>	<u>Display</u>
Floating dollar	I7M	6148	\$6148
Non-Floating dollar	I7N	5432 \$	5432
Combined Options	I5CB	-61874	(61,874)

All of the options may be specified in any order.

D Decimal number with decimal places $\text{length} \leq 15$

The data is composed of the digits 0 to 9 plus optionally a decimal point position. The values are stored internally in double precision floating point notation. If a decimal point is to be printed the format contains the point and the number of places after it.

Example:

D8.2 e.g. 3187.54
D8 e.g. 416

P Packed decimal length $\text{length} \leq 15$

The data is composed of the digits 0 to 9 plus optionally a decimal point position. Internal packed decimal storage is used. If a decimal point is to be printed the format contains the point and the number of places after it.

Example:

P9.3 e.g. 4167.368
P7 e.g. 617542

All of the edit options are available (See Integer Example).

USAGE FORMAT (cont'd)

F Decimal number with decimal places \leq length \leq 6

Similar to type D but because only 4 bytes are used internally (floating point single precision) the length of the values cannot exceed 6 digits. Hence, it is useful for small values.

Example:

F5.1 e.g. 614.2
F4 e.g. 318

All of the edit options are available. (See Integer Example).

A Alphanumeric data \leq length \leq 255

Field values which are names and textual material are assigned a type of 'A'. The length, or number of characters composing the data may be between 1 and 255 characters.

All of the letters, digits and special characters are stored with this format.

Example:

A115
A2
A24

Alphanumeric data can be printed under the control of a pattern which is supplied at run time. For instance, if a product code is to be displayed in parts, e.g.

PRODCODE/A11=EDIT (PRODCODE, '999-999-999') ;

If the value is 716431014 it will be displayed as 716-431-014. (See Report Preparation, Edit Function).

Changes:

The length part of I, F, D, and P formats may be changed. The decimal part of the length may be changed for I or D but not P data. The length of A format data may not be changed. The edit options may be changed at any time in a format.

USAGE FORMAT (cont'd)

Dates: Data fields which represent dates can be specified in a variety of ways which recognize their special nature. The elements of a date are the year, month, and day. If all three of these factors are present then the date has 6 digits and the USAGE can be:

I6MDY	display is MM/DD/YY	04/21/76
I6YMD	display is YY/MM/DD	76/04/21
I6DMY	display is DD/MM/YY	21/04/76

Date Translation: The numerical month number can be translated to the corresponding month name by adding the letter 'T' (translate) to the format. For instance:

I6MTDY ,	052176	is displayed as	MAY 21 76
I4MTY	0676	is displayed as	JUN 76
I2MT	07	is displayed as	JUL

Note that if the date represents only the month number then it has two digits and I2MTDY will display the value 4 as APR, etc. This is particularly useful in reports where columns or rows are sorted by month. They will then appear in correct calendar order, i.e. JAN, FEB, MAR, etc. because the sorting is based on the numerical values.

Two other options are available for processing dates in various situations. These are used in the Define Command mode and allow:

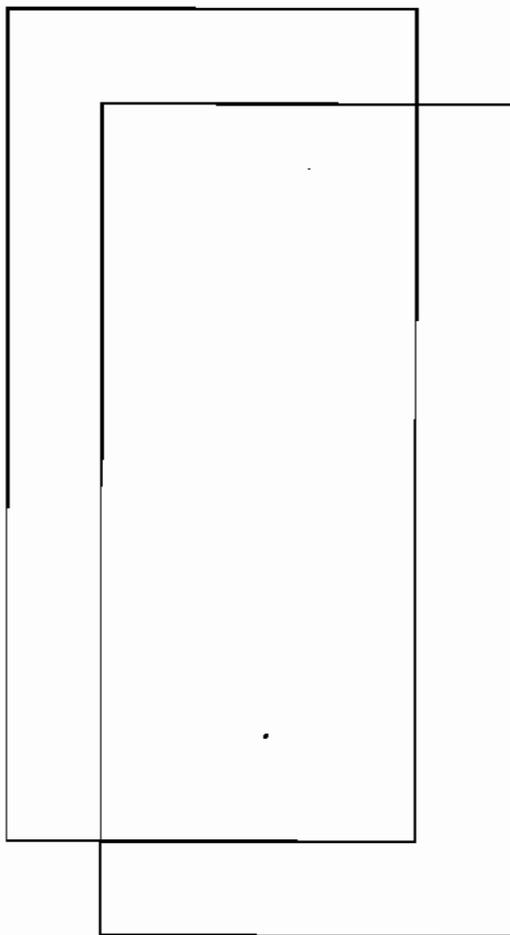
1. The duration in days between two dates to be computed. The function is called YMD ().
2. An increment to be added to a date and the resulting new date computed. For instance, 10 days added to Oct 22 is Nov 1. The function is called AYMD () (add, Year, Month, Day). (See Report Preparation - Special Functions).

Changes: The choice of edit options may be changed at any time.

<u>REFERENCE GUIDE</u>	
<u>ATTRIBUTE</u>	
FILENAME	A name of from 1 to 8 characters to be used to identify the file. The name may not contain embedded blank spaces.
SUFFIX	The SUFFIX for a FOCUS file is 'FOC'. External, non FOCUS, files use a SUFFIX to identify the type of data format or system under which the data was created. For example, a SUFFIX of COM is used to denote a free format comma delimited external data file.
SEGNAME	A name of 1 to 8 characters to be used to identify a record segment. Names do not have to be unique for segments in different files. However, since record segments can be shared between files, and the segment name is the common link it is a recommended procedure to use unique names.
SEGTYPE	The type of data organization of the record segment.
	blank Records ordered in the order of submission.
	S Records maintained in low-to-high sort order using the first data on the segment as the sequence key.
	SH Records maintained in high-to-low sort order using the first data field on the segment as the sequence key.
	KU The record segment is described in another file (it has the same SEGNAME) and is to be used in this file. A key field in this file retrieves a unique match in the other file.
	KM The record segment is described in another file (it has the same SEGNAME). A key field in this file retrieves all matches in the other file.
	KL The record segment is described in another file. It is retrieved through linkages established when its parent record is retrieved. The parent is a KM record, or KU record, or KL record.
	U The record segment can only occur once (unique), for each parent. More than one occurrence is rejected.
	KLU Same as KL (Keyed through linkage) except only one unique record is to be retrieved through the linkage.

<u>REFERENCE GUIDE</u> (cont'd)																									
<u>ATTRIBUTE</u>																									
PARENT	The name of the parent segment of which this segment is a descendent. It can be left blank if a single path file is being described.																								
CRKEY	When SEGTYPE=KU (keyed unique) or KM (keyed multiple) it is the name of the data field which is to be used as the common retrieval key.																								
CRFILE	The name of the cross-reference file in which the cross-referenced KU or KM segment is described.																								
FIELDNAME	From 1 to 12 characters to identify a field. The names should be unique within a file.																								
ALIAS	From 1 to 12 characters used as an alternate name to identify the field. It is recommended that names with no embedded blanks or special characters be used.																								
USAGE FORMAT	The type, length, and print edit options of the field.																								
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;"><u>Name</u></th> <th style="text-align: left;"><u>Types</u></th> <th style="text-align: left;"><u>Length</u></th> <th style="text-align: left;"><u>Range</u></th> </tr> </thead> <tbody> <tr> <td>Integer</td> <td>I</td> <td>n</td> <td>1<n<9</td> </tr> <tr> <td>Floating Pt.</td> <td>F</td> <td>n.m</td> <td>1<n<7 0<m<7</td> </tr> <tr> <td>Double Prec.</td> <td>D</td> <td>n.m</td> <td>1<n<16 0<m<16</td> </tr> <tr> <td>Packed Decimal</td> <td>P</td> <td>n.m</td> <td>1<n<16 0<m<8</td> </tr> <tr> <td>Alphanumeric</td> <td>A</td> <td>n</td> <td>1<n<256</td> </tr> </tbody> </table>	<u>Name</u>	<u>Types</u>	<u>Length</u>	<u>Range</u>	Integer	I	n	1<n<9	Floating Pt.	F	n.m	1<n<7 0<m<7	Double Prec.	D	n.m	1<n<16 0<m<16	Packed Decimal	P	n.m	1<n<16 0<m<8	Alphanumeric	A	n	1<n<256
<u>Name</u>	<u>Types</u>	<u>Length</u>	<u>Range</u>																						
Integer	I	n	1<n<9																						
Floating Pt.	F	n.m	1<n<7 0<m<7																						
Double Prec.	D	n.m	1<n<16 0<m<16																						
Packed Decimal	P	n.m	1<n<16 0<m<8																						
Alphanumeric	A	n	1<n<256																						

<u>REFERENCE GUIDE</u> (cont'd)																									
<u>ATTRIBUTE</u>																									
USAGE FORMAT (cont'd)	<table style="width: 100%; border: none;"> <thead> <tr> <th style="text-align: center;"><u>Edit Options</u></th> <th style="text-align: center;"><u>Action</u></th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">C</td> <td>Comma edit</td> </tr> <tr> <td style="text-align: center;">L</td> <td>Print Leading Zeros</td> </tr> <tr> <td style="text-align: center;">N</td> <td>Dollar sign non-floating</td> </tr> <tr> <td style="text-align: center;">M</td> <td>Dollar sign floating</td> </tr> <tr> <td style="text-align: center;">B</td> <td>Bracket negative values</td> </tr> <tr> <td style="text-align: center;">R</td> <td>Credit CR negative values</td> </tr> <tr> <td style="text-align: center;">S</td> <td>Suppress print of zero, print blank</td> </tr> <tr> <td style="text-align: center;">MDY</td> <td>Month-Day-Year</td> </tr> <tr> <td style="text-align: center;">YMD</td> <td>Year-Month-Day</td> </tr> <tr> <td style="text-align: center;">DMY</td> <td>Day-Month-Year</td> </tr> <tr> <td style="text-align: center;">T</td> <td>Translate month with MTDY YMTY DMTY MT</td> </tr> </tbody> </table> <p>All options may be specified in any order. Both options M and N automatically imply option C. Format type D also automatically implies option C. The month translate option T can be specified anywhere, not just following the letter M.</p>	<u>Edit Options</u>	<u>Action</u>	C	Comma edit	L	Print Leading Zeros	N	Dollar sign non-floating	M	Dollar sign floating	B	Bracket negative values	R	Credit CR negative values	S	Suppress print of zero, print blank	MDY	Month-Day-Year	YMD	Year-Month-Day	DMY	Day-Month-Year	T	Translate month with MTDY YMTY DMTY MT
<u>Edit Options</u>	<u>Action</u>																								
C	Comma edit																								
L	Print Leading Zeros																								
N	Dollar sign non-floating																								
M	Dollar sign floating																								
B	Bracket negative values																								
R	Credit CR negative values																								
S	Suppress print of zero, print blank																								
MDY	Month-Day-Year																								
YMD	Year-Month-Day																								
DMY	Day-Month-Year																								
T	Translate month with MTDY YMTY DMTY MT																								
ACTUAL FORMAT	This element may be left blank, but is necessary when non-FOCUS files are described. (See Describing External Files).																								
FIELDTYPE	If this element is set to 'I' it means that an index is to be maintained of the location of all occurrences of all values of the field. The presence of this index allows other segments from other files to link to this segment. A segment may be used as a SEGTYPE KU, or KM in another file only if an index is available.																								



F O C U S

USER'S MANUAL

INFORMATION BUILDERS INC.
254 WEST 31st STREET
NEW YORK, N.Y. 10001
(212) 736-4433

TABLE OF CONTENTS

GENERAL CONCEPTS4-01
Tasks Performed by the FOCUS Transaction Processor4-02
Syntax of the MODIFY Command4-02.1
Basic Processes4-02.3
DESCRIBING TRANSACTIONS4-03
TASK 1 - Describing Transactions (Fixed Format)4-03
FORM Subcommand4-04
Embedded Blanks4-06
Binary Data Values4-07
Describing Repeating Groups (Fixed Format)4-08
Describing Data Value Which May Not Be Present (Conditional Data)4-09
TASK 1 - Describing Transactions (Free Format)4-10
Leading Blanks4-11
Data Containing Special Characteristics4-11
FREEFORM Subcommand4-13.1
TASK 2 - Validating Transactions4-14
TASK 3 - Calculating New Values4-16
TASK 4 - Matching Transactions to Data Base Records4-17
MATCH * - Including Records on Detail Levels of a File4-18
TASK 5 - Action Upon Matching or Not Matching4-19
Updating With Free Format Transactions4-21
Default Conditions4-22
Control of Duplicate Records ON MATCH INCLUDE4-23
MATCH COMPUTE4-24
TASK 6 - Error Logs and Error Messages4-25
TASK 7 - Source of Transactions4-27
START4-29
STOP4-29
Prompting for Transaction Values4-30
Correcting a Prior Response4-31
Typing Ahead4-32
Prompting for Groups of Fields4-33
SPECIAL TOPICS4-35
Annotating MODIFY Procedures4-35
Erasing a FOCUS File4-36
RECOVERY4-37
File Status4-37
SUMMARY OF SUBCOMMANDS4-39
Meaning and Usage4-39
ERROR MESSAGES4-43
File Maintenance Error Messages4-43

GENERAL CONCEPTS

In order for information in a FOCUS data base to be changed new information has to be submitted to either:

- . change the values of existing items in existing records
- . add new records of information
- . delete old records

The new information is called a 'transaction'. These transactions may be entered in several ways; They may be;

- . typed directly on a communication terminal into the data base
- . stored in a temporary disk file, or tape file for later entry into the data base
- . entered in response to interactive prompting for information

Regardless of how the transactions are entered there are 7 elements of information about them which must be known.

FOCUS provides a simple to use language for describing these 7 elements. The language substitutes common defaults when information is not provided thus reducing the users effort to a minimum. The purpose of the transaction processing language is to replace the need for writing a computer program in order to modify information in a FOCUS data base.

The transaction processor is entered by typing the FOCUS command MODIFY followed by the name of the file to be changed. For instance,

```
MODIFY FILE CARS
```

The system then responds by typing ENTER SUBCOMMANDS:

There are 12 subcommands which are available for accomplishing the 7 descriptive tasks about the transactions. In most cases only a few of these subcommands are needed. The tasks and subcommands are:

Tasks Performed by the FOCUS Transaction Processor

Task 1 Describe the physical format or layout of the fields on the transaction.

Subcommands: FIXFORM (FORM)
 FREEFORM

Task 2 Describe the validation criteria for accepting a transaction.

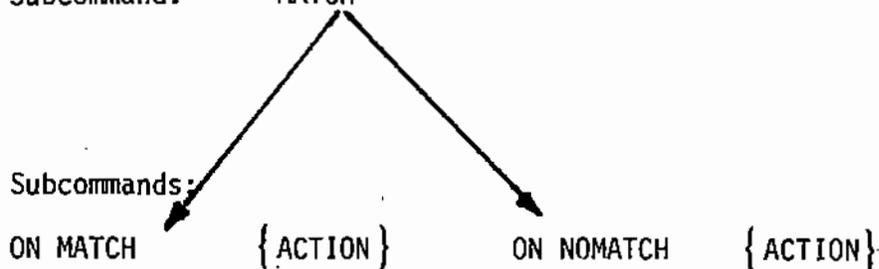
Subcommands: VALIDATE

Task 3 Describe calculations or modifications to be made to the data on the transaction.

Subcommand: COMPUTE

Task 4 Describe how the transaction is to match data base records.

Subcommand: MATCH



Task 5 Describe the action which is to occur when transaction records match or do not match data base records.

Subcommands:

ON MATCH	$\left\{ \begin{array}{l} \text{UPDATE} \\ \text{REJECT} \\ \text{DELETE} \\ \text{COMPUTE} \\ \text{INCLUDE} \end{array} \right\}$	ON NOMATCH	$\left\{ \begin{array}{l} \text{INCLUDE} \\ \text{REJECT} \\ \text{COMPUTE} \end{array} \right\}$
----------	---	------------	---

Task 6 Describe how errors are to be treated as well as any audit information for accepted transactions.

Subcommand: LOG

Task 7 Describe where the source of the transactions come from.

Subcommands: START
 STOP
 DATA
 PROMPT

Syntax of the MODIFY Command

All options of the FOCUS transaction processing system are invoked through the use of the FOCUS MODIFY command. After the command follows the name of the file to be modified. For example, MODIFY FILE CARS. The transactions which are to modify the data file may come from disk or tape files, or may be typed directly on the terminal.

. TRANSACTIONS ON DISK OR TAPE:

When the transactions come from disk or tape files the procedure is terminated by the word END on a line by itself.

Example: Transactions on a disk file

invokes processor	→	MODIFY FILE CARS
layout of transactions	→	FIXFORM COUNTRY/10 X2 CAR/10 MODEL/20 RCOST/8
actions to be taken	→	{MATCH COUNTRY CAR MODEL ON MATCH UPDATE RCOST ON NOMATCH REJECT
source of transactions	→	DATA ON NEWCOSTS
end of procedure	→	END

TRANSACTIONS TYPED 'LIVE':

When the transactions are to be typed 'live' on the terminal by the operator the subcommand word DATA is the last subcommand processed. The terminal then opens to receive data and the operator types the transactions (usually in either free format or via prompting assistance) and the word END on a line by itself is used to terminate the procedure after the last transaction is typed.

Syntax of the MODIFY Command (cont'd)

Example: Transactions typed 'live' in free format

invoke processor	→	MODIFY FILE CARS
actions to be taken	→	{ MATCH COUNTRY CAR MODEL BODY
		{ ON NOMATCH INCLUDE
		{ ON MATCH REJECT
start of data	→	DATA
transaction data	→	{ COUNTRY=ITALY, CAR=FIAT, MODEL=128 2 DOOR
		{ BODY=COUPE, SEATS=5, RCOST=4150, \$
		{ MODEL=128 4 DOOR, BODY=SEDAN, STATS=6,
		{ RCOST=4355, \$
end of data and procedure	→	END

Basic Processes

The three basic file maintenance processes are:

- . input of new segments
- . change of values of existing segments
- . deletion of existing segments

These can be combined in one procedure which then performs compound processes. The examples below illustrate each of the basic processes.

Example: Input of new segments

for rejected records	→	FILEDEF NOGOOD DISK REJECT DATA LRECL 80 RECFM F
		MODIFY FILE CARS
layout of transactions	→	{ FIXFORM COUNTRY/10 X2 CAR/10 MODEL/20
		{ FIXFORM BODY/5 SEATS/3 X4 RCOST/6 DCOST/6
match all fields	→	MATCH *
if new data, include it	→	ON NOMATCH INCLUDE
if not new, reject it	→	ON MATCH REJECT
keys a file of rejects	→	LOG REJECTS ON NOGOOD
source of data	→	DATA ON NEWIN
		END

Example: Update of Existing Data Values

* see note		DEFINE FILE CARS
establish validation criteria	→	SEATEST=SEATS EQ 2 AND BODY EQ COUPE
		OR STATS GT 3 ;
		END
		MODIFY FILE CARS
describe layout of transaction	→	{ FIXFORM CAR/10 X2 COUNTRY/10 MODEL/20
		{ FIXFORM X4 BODY/5 SEATS/3 RCOST/6
invoke validation tests	→	VALIDATE SEATEST
specify action updates	→	{ MATCH CAR COUNTRY MODEL BODY
		{ ON MATCH UPDATE SEATS RCOST
		{ ON NOMATCH REJECT
source of data	→	DATA ON UPVAL
		END

* See Section 1 Report Preparation, Calculations for all of the types of calculation and tests which may be defined.

Basic Processes (cont'd)Example: Deleting Segments

```
match—————▶ MODIFY FILE CAR
                  MATCH COUNTRY
actions————▶ {ON NOMATCH REJECT
                  {ON MATCH CONTINUE
match—————▶ MATCH CAR
actions————▶ {ON NOMATCH REJECT
                  {ON MATCH CONTINUE
match—————▶ MATCH MODEL
actions-deletion▶ {ON MATCH DELETE
                  {ON NOMATCH REJECT
transactions follow▶ DATA
                  .
                  .
                  .
                  .
                  END
```

TASK 1Describing Transactions (Fixed Format)

Fixed format transactions are those in which field values occupy the same position in each transaction record. In order to describe the fixed layout two pieces of information are necessary.

- . First - The names of the fields must be given and
- . Second -the columnar position of the field in the record.

The FORM subcommand is used for these purposes. As many lines of the FORM subcommand as are needed to describe the layout of the transaction record can be submitted.

FORM Subcommand

The FORM subcommand is used to specify the names of the data fields and the number of characters to be processed for each data field on the transaction.

The basic syntax is:

FORM fieldname/length fieldname/length...

The fieldname is separated by a slash from the number of characters to process on the transaction to obtain the value for the field. Each set of name-lengths is separated by a blank from the next set of name-lengths. For example:

FORM CUSTOMER/10 MONTH/2 UNITS/4 AMOUNT/5

This indicates that the transaction record has four data fields, the first is 10 characters in length, the second 2 characters, the third 4 characters, and the last is 5 characters. If any characters on the transaction record are to be skipped then an 'X' followed by the number of characters to skip is used. For example:

FORM X3 CUSTOMER/10 MONTH/2 X1 UNITS/4

The first 3 characters are skipped, and later 1 character is skipped. It is possible to skip backwards as well as forwards. This permits a single transaction field to be used as the value for several data file fields. For example, to add to a year-to-date total, and also be input as a detail value. Hence,

FORM YRTODATE/4 X-4 UNITS/4 AMOUNT/8

will process 4 characters then back up and take the same value again and use it as if it were a different data item. A partial split of transaction field can also be used. For example, a prefix on a data field may be stored as a separate item, as well as the whole data item. e.g.

FORM PRODUCT/8 X-8 CODE/2 X6 FACTOR/5

This uses all 8 characters as one data value, then backs up to the beginning of the field and uses the first 2 characters, then skips over the remaining 6 to continue with the other values.

Any number of entries separated by blanks may be provided on a given FORM line. If more lines are needed then the entries follow each other in a stream of FORM lines. The following two sets are identical in meaning.

Set 1

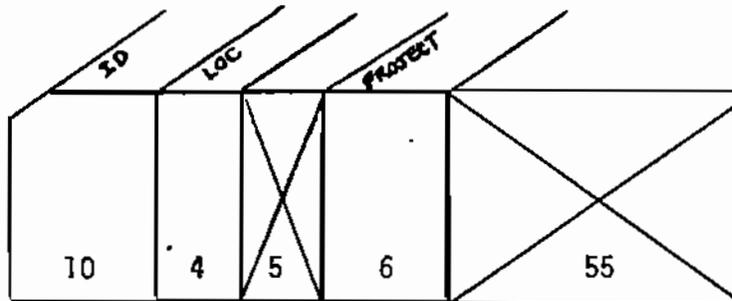
FORM MONTH/2 CODE/3 X40 UNITS/4 FACTOR/6

Set 2

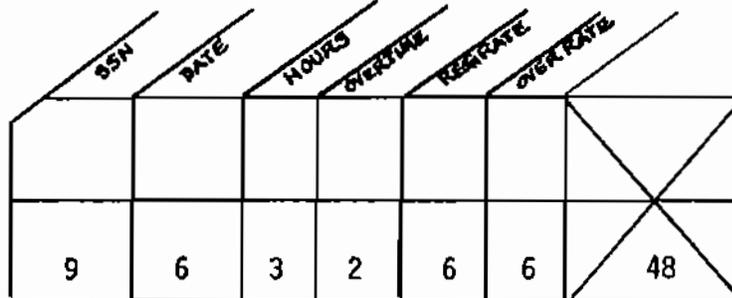
FORM MONTH/2 CODE/3 X40
 FORM UNITS/4
 FORM FACTOR/6

Example: Describing a Fixed Layout Transaction Record

A transaction composed of two 80 column key punch cards contains 9 data fields. Note the use of the skip 'x' to pass over columns which are not used.



FORM ID/10 LOC/4 X5 PROJECT/6 X55



FORM SSN/9 DATE/6 HOURS/3

FORM OVERTIME/2 REGRATE/6 OVERRATE/6 X48

Embedded Blanks:

If any fieldname on the FORM subcommand contains an embedded blank then either use its alternate alias name, or shortest unique truncation, or enclose the element in single quotes. For instance:

```
FORM 'RETAIL COST/9' MODEL/18
```

single quotes embedded blank

Example: File Update Using Fixed Format Transaction

```
MODIFY FILE CARS
FORM COUNTRY/10 CAR/18 'MODEL NAME/20' TYPE/8
FORM SEATS/2 X10 RCOST/8 DCOST/8
MATCH COUNTRY CAR MODEL TYPE
ON MATCH UPDATE SEATS RCOST DCOST
ON NOMATCH REJECT
LOG NOMATCH ON BADRECS
DATA ON UPFILE1
END
```

Binary Data Values

If data values on the transaction already exist in internal format such as binary integer, packed decimal, zoned decimal, or floating point notation then the length sub-parameter after the fieldname on the FORM subcommand must state the type of data value.

The types of internal data are:

<u>Letter</u>	<u>Type</u>
P	Packed decimal (length in packed digits plus sign)
I	Binary Integer (2 bytes)
D	Double precision floating point (8 bytes)
F	Single precision floating point (4 bytes)
Z	Zoned decimal

Example: Reading Binary Data

FORM COUNTRY/10 RCOST/P8 SEATS/I4 CODE/Z5

↑
Data is already in
packed decimal

↑
Data is in
zoned decimal

Describing Repeating Groups (Fixed Format)

A common situation is the occurrence of sets of fields more than once in the transaction. For instance, a record which contains a product name, then sales in each of 12 months. The set MONTH and AMOUNT occurs 12 times in this example.

Sets are indicated on the FORM subcommand by enclosing the fields in the set between parentheses, and placing the number of occurrences before the open parenthesis. For example:

```
FORM PRODUCT/10 AREA/6 12 (MONTH/2 X4 AMOUNT/6)
```

Note that the length of this record is:

$10 + 6 + 12 \times (2 + 4 + 6)$ or 160 characters

Multiple groups can be specified but they cannot be nested.

Example: Input of Repeating Groups

```
MODIFY FILE CARS
MATCH COUNTRY CAR MODEL
ON NOMATCH INCLUDE
MATCH WARRANTY
ON NOMATCH INCLUDE
FORM COUNTRY/10 3 (CAR/10 MODEL/20) 2 (WARRANTY/30)
DATA ON MYNEW
END
```

Note: Number of occurrences

Describing Data Value Which May Not Be Present (Conditional Data)

Data values which may occur on one transaction but not another are said to be conditionally present. If they occur, they take part in the input, or update process, else they are ignored. The test for occurrence is a completely blank field.

Two major uses are:

1. Input of a variable number of groups.
The length specification on the FORM line for each field in the group is preceded by the letter 'C'.

Example: Conditional Presence of Input Data

FORM PRODCODE/8 X3 12 (MONTH/C2 X1 SALES/C6)

conditional

Sample data might appear as:

SP40721X...01..1465.02...467.03..5148
RV431674...01..6400.

The physical transaction records in the example may have from 0 to 12 groups. Each group which is accepted is counted as one logical transaction. Hence, the two physical records above would be counted as 4 transactions.

2. Update of a variable list of fields.
The length specification for each field which may not be present is preceded by the letter 'C'.

Example: Update of a Variable Set of Fields

```

MODIFY FILE CARS
FORM COUNTRY/10 CAR/15 MODEL/15 BODY/8
FORM MPG/C4 WEIGHT/C5 LENGTH/C5
FORM WBASE/C5
MATCH COUNTRY CAR MODEL BODY
ON MATCH UPDATE MPG WEIGHT LENGTH WBASE
ON NOMATCH REJECT
DATA ON NEWVALS
END
    
```

In the example, some transactions might contain all of the fields to update, e.g. MPG, WEIGHT, LENGTH, etc. Others might contain only WEIGHT, or some partial set of the total fields available for UPDATE.

TASK 1Describing Transactions (Free Format)

In a free format transaction the operator types the identity of the data field along with its data value. Typing is not restricted to fixed columns on a line, and each line may be entirely different. The data fields can be identified in 4 ways;

1. Fieldname
2. Synonym name
3. Numerical position in data file dictionary
4. Shortest unique truncation

Example: Equivalent Methods

Assume a data file with the following 5 data fields

<u>Number</u>	<u>Name</u>	<u>Synonym</u>
1	PROD-TYPE	PRT
2	PROD-NAME	PNAME
3	AREA	AR
4	AMOUNT	AMT
5	FACTOR	FAC

- (a) PROD-TYPE=ALY, AREA=TRUCKS, AMOUNT=1000,\$
- (b) PRT=ALY, AR=TRUCKS, AMT=1000, \$
- (c) 1=ALY, 3=TRUCKS, 4=1000, \$
- (d) 1=ALY
3=TRUCKS, 1000, \$

All of the above four are equivalent ways to type the same transaction. (Assuming PROD-TYPE is field 1 in the file dictionary, etc.).

Note that:

- (a) Data values, or fieldname-value pairs are separated from each other by commas.
- (b) The end of each transaction is terminated by a comma followed by a dollar sign, '\$'. The transaction may continue to any number of lines, and the last line contains the terminator.
- (c) Once a data field has been mentioned on one transaction it doesn't have to be re-set in value, and the old value will persist until it is changed making it easy to enter repetitive information.

Leading Blanks

Blank spaces before or after a data value have no effect and are removed. Hence, typing can begin anywhere on a line. For example,

2 = ALLOY

and

2=ALLOY

will produce the same result.

Data Containing Special Characters

When a data value contains a special character such as a comma, or dollar sign, or leading blank spaces are desired, then the value should be enclosed between single quote marks. For instance,

LOCATION='ORANGE, NEW JERSEY'

The embedded comma is ignored and becomes part of the data value.

Example: Performing Both an Update and Input Using Free Format Data

The purpose of this procedure is to change the RETAIL COST and DEALERS COST for all cars and models which are provided. If a given model is not found in the data base then a new record is included. Only the inclusion of additional models are allowed in the data base, but new countries, or cars are not permitted, any mismatch of country or car causes the transaction to be rejected. Data is being entered live using a free comma delimited format.

Definition of test condition → DEFINE FILE CARS
 COST-TEST=DCOST LT RCOST AND RCOST GT 0 ;
 END

Command → MODIFY FILE CARS
 MATCH COUNTRY CAR
 ON NOMATCH REJECT
 MATCH MODEL TYPE
 ON MATCH UPDATE RCOST DCOST
 ON NOMATCH INCLUDE
 VALIDATE COST-TEST

Start to enter data → DATA
 COUNTRY=ITALY, CAR=FIAT, MODEL=128 3 DOOR, WAGON
 DCOST=2718, 3207, \$
 MODEL=131 4 DOOR, SEDAN, DCOST=3382, 4092, \$
 .
 .
 etc.
 .

No more data → END

Example: Carry-over of Values

```
MODIFY FILE SAMPLE
MATCH PRT AREA AMT
DATA
PRT=ALY, PNAME=ALLOYS
AREA=TRUCKS, 1000,.50, $
AREA=CARS, 500,.42, $
AREA=BUSES, 1100,.51, $
END
```

Note that the data values for PROD-TYPE and PROD-NAME were supplied in the first transaction and since no other values are supplied they are still active for the second and third transaction.

A computational value can be set to zero, and alphanumeric data to blank by using extra comma separators. For example,

```
1=STL,,CARS,,,$
```

Fields 2, 4 and 5 are set to zero or blanks as needed.

FREEFORM Subcommand

Data values which are entered in free format are separated from each other by commas. The default assumption is that each comma represents a jump to the next field in the order that the fields are described in the MASTER dictionary. Hence, a transaction record composed of the following data would represent 7 data fields:

	A,	B,	C,	,	,	V,	W,	\$
Fields:	↑	↑	↑	↑	↑	↑	↑	↑
	1	2	3	4	5	6	7	

Note that after 'C', the third field, the fourth field and the fifth field which have no data value are still counted because of the commas. This transaction record could have been presented as:

A, B, C, 6=V, W, \$

The break because of the absence of fields 4 and 5 is covered by explicitly stating 6= (or fieldname =).

When the natural order of the fields is not a convenient way to type the values because of breaks, or because a different sequence is easier, i.e. 3, 2, 1 rather than 1, 2, 3, then the FREEFORM subcommand is used.

The FREEFORM subcommand specifies the order in which the fields are presented on the transaction. The commas are interpreted as the start of a new field, not one greater than the last, but as the next one on the FREEFORM list. The syntax of the command is:

FREEFORM field field field field

FREEFORM Subcommand (cont'd)

The arguments of the subcommand is a list of field names in the order these fields will be presented.

Example: Free format Data

```

MODIFY FILE CARS
* FOUR FIELDS PROVIDED
* THREE TO LOCATE RECORD
* ONE TO BE UPDATED
FREEFORM CAR COUNTRY MODEL MPG
MATCH CAR COUNTRY MODEL
ON MATCH UPDATE MPG
ON NOMATCH REJECT
DATA
FIAT, ITALY, 128 2 DOOR, 31, $
MASERATI, ITALY, FORA 2 DOOR, 29, $
DATSUN, JAPAN, 260Z, 33, $
END

```

Note that in the file description the fields CAR COUNTRY MODEL and MPG have the order of 2, 1, 3, 12.

Data values may be absent from a transaction in which case a break is denoted by a field being identified by its name. i.e.

```

FIAT, ITALY, 128 2 DOOR, 31, $
MODEL=128 4 DOOR AUTO 24, $

```

Carryover of CAR and COUNTRY

Transactions should be ended by a terminator character of a '\$'. If there is an error detected in the transaction then the end of the transaction is the first terminator found.

When the FREEFORM subcommand is used to override the natural order of field occurrence one important rule must be observed.

- Fields which are not mentioned on the FREEFORM list cannot be supplied. If a transaction record contains an extra comma, or a fieldname which is not on the list, the transaction will be rejected.

TASK 2Validating Transactions

A validation test is a defined calculation on the values in a transaction record. If the resulting value is zero, the transaction is rejected as invalid, or else it is accepted. The validation tests are stored for future use using the FOCUS DEFINE command. (See Section 1 Report Preparation Calculations). The particular tests which are wanted in a given procedure are then invoked by naming them on the VALIDATE subcommand.

Step 1 Define the validation criteria, e.g.

```
DEFINE FILE CARS
SEAT-TEST=IF SEATS LT 0 OR SEATS GT 9 THEN 0 ELSE 1 ;
COST-TEST=IF RCOST LT DCOST THEN 0 ELSE 1 ;
END
```

Step 2 Name the tests to use in the Data Management procedure

```
MODIFY FILE CARS
:
:
VALIDATE SEAT-TEST COST-TEST
:
:
DATA
```

When a data value must be one of a small list in order to be valid the DECODE function can be used. For example:

```
GOODTYPE/I1=DECODE BODYTYPE (SEDAN 1 CONVERTIBLE 1)
'HARD TOP' 1 COUPE 1 WAGON 1 ELSE 0) ;
```

If the BODYTYPE is not one of 5 named the value of GOODTYPE takes on the default of zero and the record will be rejected.

BLANK PAGE

TASK 3Calculating New Values

From the data values on the transaction new data values can be computed or constants inserted. These can define missing values on the transaction or modify submitted values. Calculations for example, can be used to scale a number by multiplying by a constant, or to add a constant date to all transactions submitted on that date.

Like the VALIDATE subcommand the calculations are defined for future use in the DEFINE mode (See Section 1 FOCUS Users Manual, Calculations). The particular calculations wanted for a given procedure are then invoked by naming them on the COMPUTE SUBCOMMAND.

Example: Calculating Values

Step 1 Defining the calculation

```
DEFINE FILE CARS
RCOST=IF COUNTRY EQ 'ITALY' THEN RCOST/600 ELSE RCOST ;
BODYCODE/I1= ;
BODY=DECODE BODYCODE (1 SEDAN 2 CONVERTIBLE 3 ROADSTER
4 WAGON 5 COUPE ELSE OTHER) ;
DATE=760421 ;
END
```

Step 2 Invoking calculations

```
MODIFY FILE CARS
FORM COUNTRY/10 CAR/18 MODEL/18 BODYCODE/1
MATCH COUNTRY CAR MODEL
ON NOMATCH INCLUDE
ON MATCH UPDATE BODY RCOST DATE
COMPUTE BODY RCOST DATE
DATA ON EXTERN
END
```

Note in this example that the field named BODYCODE is not a data base field. It occurs only on the transaction. Based on its value a corresponding value is found for a data base field named BODY.

TASK 4Matching Transactions to Data Base Records

The basic operation during transaction processing is to match some values from a transaction to corresponding values in the data base and take action depending upon whether there is a complete match, partial match, or no match.

The designation of the fields to match are provided on the subcommand MATCH. The names of the fields are listed one after the other separated by blanks. (if a name contains an embedded blank enclose it in single quotes).

As many lines of MATCH subcommand as are needed may be submitted.

e.g.

```
MATCH COUNTRY MODEL CAR  
MATCH SEATS TYPE
```

Note that fieldnames can always be referenced by either their full fieldname, alternate alias name, or shortest unique truncation.

Hence: MATCH COU MO CAR SE TY
may be equivalent to the above

After the MATCH subcommand the actions to be taken are specified through the ON MATCH and ON NOMATCH subcommands.

Including Records on Detail Levels of a FileMATCH *

The upper levels of a file frequently contain important keys used to locate records while lower levels often contain the bulk of the data fields. A simple way to refer to all the fields on these levels is to place a MATCH * subcommand after the main control information. (The '*' is a frequently used shorthand symbol for 'all'). The placement of this subcommand in the MODIFY control stream is usually last because it is effective only for segments which have not been mentioned. That is, those for which no other information has been provided.

Example: Use of MATCH *

```
MODIFY FILE CARS
FORM COUNTRY/10 CAR/10 MODEL/20 BODY/5 RCOST/6
FORM DCOST/6 LONG/5 WEIGHT/5 WBASE/5 MPG/4
FORM WARRANTY/40
MATCH COUNTRY
ON MATCH REJECT
MATCH *
ON NOMATCH INCLUDE
DATA ON NEWCARS
END
```

TASK 5Action Upon Matching or Not Matching

Either a transaction matches a data base record, in which case four actions can occur, or it doesn't match and two actions can occur.

ON MATCH If a transaction matches all of the fields named on the prior MATCH subcommand then the four actions which can occur are:

1. ON MATCH REJECT

The transaction is considered a duplicate (at least to the point of match) and is therefore to be rejected.

2. ON MATCH UPDATE fieldname fieldname...etc.

A data base record matching the keys named on the prior subcommand, if found, is to have the following list of fields updated. (Several lines each starting with the word UPDATE may be used if more than 1 line is needed).

3. ON MATCH DELETE

From the point of match a record segment is deleted, and all dependent records and all references to the deleted fields in all indexes are removed.

4. ON MATCH COMPUTE name name...etc.

The items named are executed. The items may refer to the transaction data field separately from the data base fields and changes are performed on the records based on the logic in the procedures.

5. ON MATCH INCLUDE

Even if a transaction matches a data base record it is to be included. Possibly unique keys cannot be identified, or the order of entry identifies records.

Task 5Action Upon Matching or Not Matching (cont'd)

ON NOMATCH If a transaction does not match all of the fields named on the prior MATCH subcommand then two actions can occur.

1. ON NOMATCH REJECT

A transaction which fails to find a corresponding data base record is to be rejected. This is frequently coupled with an UPDATE, as matching records are changed, and non matching ones rejected.

2. ON NOMATCH INCLUDE

A transaction which fails to find a corresponding data base record is to be included in the data base itself. This is the basic process for record input.

Example: Input of New Records

```
MODIFY FILE CARS
FORM COUNTRY/10 CAR/18 MODEL/6 SEATS/2 TYPE/19 RCOST/6
MATCH COUNTRY CAR MODEL TYPE
ON NOMATCH INCLUDE
DATA ON NEWSTUFF
END
```

The sequence of MATCH, and ON subcommands is significant when compound processes are to be performed. For instance, a record might be rejected if it doesn't match up to a point, but from that point it may be acceptable to include it in the data base. The following example illustrates this:

Example: Partial Matching and Updating or Including

```
MODIFY FILE CARS
MATCH COUNTRY CAR
ON NOMATCH REJECT
MATCH MODEL TYPE
ON NOMATCH INCLUDE
ON MATCH UPDATE RCOST
DATA
```

Updating With Free Format Transactions

When the data values are provided in free format (comma delimited) each transaction may have a different list of fields to be updated. If a field which is mentioned on the UPDATE subcommand is not given a value in a transaction then no change to the corresponding data base value occurs. This is the counterpart to the Conditionally Present option in fixed formats.

Example: Updating in Free Format

```
MODIFY FILE CARS
MATCH COUNTRY CAR MODEL
ON MATCH UPDATE MPG WBASE WEIGHT
ON NOMATCH REJECT
DATA
COUNTRY=JAPAN, CAR= DATSUN, MOD=B210, MPG=33,
(1) → WEIGHT=2050, $
(2) → MOD=610, MPG=27, WBASE=98.4, $
(3) → MOD=710, WBASE=96.5, $
END
```

- (1) Transaction 1 - Only MPG and WEIGHT is changed.
- (2) Transaction 2 - Only MPG and WBASE changed.
- (3) Transaction 3 - Only WBASE changed.

Default ConditionsMATCH

If no MATCH subcommand is provided, then all the fields which are processed will be used by default as the matching conditions. This is useful for initial data entry.

Example:

```
MODIFY FILE CARS
FORM COUNTRY/10 CAR/10 MODEL/20 SEATS/2 RCOST/5
FORM DCOST
DATA ON NEWSTUFF
END
```

When a file has data in it, then it is not recommended that the default be used. It is a better practice to specifically select the fields to be matched.

ON NOMATCH INCLUDE

Should a transaction not match a data base record and no instruction provided on either match or no-match then the default is:

```
ON NOMATCH INCLUDE
```

As soon as any instruction is provided on what to do if there is a match i.e. ON MATCH UPDATE then the default changes to ON NOMATCH REJECT.

Hence, a procedure which is to both UPDATE and INCLUDE must specifically state these actions. The reason for using these defaults is that they clearly relate to three separate processes and simplifies the logic of each:

1. Pure inclusion of new records, no update; default is ON NOMATCH INCLUDE.
2. Pure update, no inclusion of new records; default is ON NOMATCH REJECT.
3. Both update and inclusion; actions must be specified.

Control of Duplicate Records ON MATCH INCLUDE

The default of the MODIFY command is to reject transactions if a duplicate exists in the data base. However, since only the data fields mentioned in the MATCH subcommand are compared to their counterparts in the data file it is important to clearly specify which parts of the transactions identify a unique record, else then can be rejected based on too small a subset of fields. In some situations the transactions may not have unique values or for other reasons all transactions are to be included in the data file. This is accomplished by the subcommand:

```
ON MATCH INCLUDE
```

If the MATCH is to encompass all of the data fields then the above subcommand should be issued in the following order.

```
MATCH *  
ON MATCH INCLUDE
```

Rejection of duplicates of some segments but not others is controlled in the usual way. Consider the following example.

Example:

On a given day several test readings of river pollution are taken; the readings may have the same values, but are all to be included. In another section of the file, the file river traffic is recorded and control for duplicates is needed.

```
MODIFY FILE RIVDATA  
MATCH RIVER  
ON NOMATCH REJECT  
MATCH DATE SHIP CARGO WEIGHT  
ON MATCH REJECT  
ON NOMATCH INCLUDE  
MATCH TEMPERATURE BACTCOUNT  
ON NOMATCH INCLUDE  
ON MATCH INCLUDE  
DATA ON DAILYRPT  
END
```

MATCH COMPUTE

The material in this section is included for reference purposes only. The COMPUTE option of MATCH is not a feature of release 1.0 of FOCUS.

The purpose of COMPUTE is to specify the logic of how transaction data values are to change values in the FOCUS data file in cases other than a pure Update by replacement. The syntax used to state the calculation is the same as used in the DEFINE mode. (See Report Preparation - Calculations). Fields from the data file and fields from the transaction are distinguished from each other by the use of a prefix of 'D' in front of the data base field. For instance D.RCOST refers to the value of the field RCOST as obtained from the data base.

Example: Cumulative adding of new value to existing value.

```
RCOST=RCOST + D.RCOST;
```

Example: Change of data base value only if it is zero.

```
RCOST=IF D.RCOST EQ 0 THEN RCOST ELSE D.RCOST;
```

Example: Change of data base value only if both transaction value is non-blank and data base value is blank.

```
ADDRESS=IF ADDRESS NE ' ' AND D.ADDRESS EQ ' ' THEN ADDRESS;
```

Example: A value in the record is used to compute new value.

```
RCOST=IF D.FACTOR GT 0 THEN D.FACTOR*RCOST ELSE RCOST;
```

Note: The action, ON MATCH UPDATE is a default calculation of field=field, data value replaced by transaction value.

TASK 6Error Logs and Error Messages

The default operation is to display all rejected transactions on the terminal and print the corresponding reason for the rejection. These reasons are:

- | | |
|------------|---|
| 1. NOMATCH | The transaction fails to match a data base record although required by the procedure. |
| 2. DUPL | The transaction matches a data base record although not required to by procedure and is therefore considered a duplicate. |
| 3. INVALID | A VALIDATE criteria test fails. |
| 4. FORMAT | A format error in the data is detected, i.e. non numeric characters in numeric field. |

The default of echoing rejects on the terminal can be changed, and the transactions can be written to log files which have been FILEDEF'ed prior to the start of the procedure.

Also all transactions accepted or rejected can be logged, or only all accepted transactions which modified the data file can be logged.

- | | |
|-----------|--------------------------|
| 5. ACCEPT | An accepted transaction. |
| 6. TRANS | All records presented. |

The syntax for specifying logging, or just error message control is:

```
LOG type ON ddname MSG {ON }
                        {OFF }
```

or if only logging is affected

```
LOG type ON ddname
```

or if only messages are affected

```
LOG type MSG {ON }
              {OFF }
```

The entire transaction record is written to any log file in the same format as it was read. Up to six log files can be active in one procedure.

Example:

```
LOG NOMATCH ON BADMAT
LOG FORMAT ON BADDAT
LOG DUPL ON GOT MSG OFF
LOG ACCEPT ON TRAIL
LOG INVALID MSG OFF
```

The log files must be given a complete FILEDEF including record length.

The TRANS file should have the same FILEDEF as the original transaction stream.

The other log files should be given a length equal to the length of the 'logical' record. For instance, if three 80 character transaction records are used to produce one logical transaction then the length of the logical file is 240 characters.

Example: Logging NOMATCH's During MODIFY Procedure

```
FILEDEF NOPE DISK NOPE VALUE (LRECL 90 RECRM FB BLKSIZE 90)
FILEDEF NEWVAL DISK NEWS DATA (LRECL 90 LRECL FB BLKSIZE 90)
.
.
MODIFY FILE CARS
FORM COUNTRY/10 X10 CAR/15 MODEL/20 X5 MPG/5 X35
LOG NOMATCH ON NOPE MSG OFF
MATCH COUNTRY CAR MODEL
ON NOMATCH INCLUDE
DATA ON NEWVAL
END
```

TASK 7Source of Transactions

Transactions can be provided in three ways.

1. Typed directly on the terminal in either free format, or in fixed format.
2. Stored in temporary files on tape, or disk. (from key punch cards, paper tape, tape cassettes, or from programs)
3. Typed directly on the terminal, but in response to prompting inquiries for each data value.

The last subcommand which must always be provided as it denotes the end of the list of subcommands, and the start of data transfer is either;

DATA

or

DATA ON ddname

When the subcommand is only the word DATA it means that the transactions will follow on the terminal.

When the subcommand is;

DATA ON ddname

where ddname is the name used in a CMS FILEDEF statement to help locate stored transactions, then the transactions are read from this file.

Example: Reading a Transaction File

```
FILEDEF MYDATA DISK TODAY UPDATE (RECFM F LRECL 100)
.
.
FOCUS
MODIFY FILE CARS
.
.
DATA ON MYDATA
```

TASK 7Source of Transactions (cont'd)

If the transaction file is given a CMS filetype of DATA and the records are all 80 characters (such as produced by the EDITOR) then no FILEDEF is required and only the CMS filename is needed.

START

The first transaction to be processed is by default the first one read. This can be changed, and the n^{th} transaction the first one processed. This allows a transaction stream to be processed in parts, or a run repeated from a 'pick-up' point. The syntax is:

START N

where N is an integer number.

The number refers to physical records, hence, if a logical record is composed of several physical records this should be multiplied into the start number. For instance, if 4 physical records of 80 characters each compose 1 logical transaction, then the 20th transaction starts at the $4 \times 20 + 1 = 81$ st physical record. Hence, START 81 would be used.

STOP

The last transaction to be processed, particularly for test purposes can be specified by:

STOP N

where N is an integer number.

This number is the last 'physical' record to be read.

Example:

```
MODIFY FILE CARS
FORM COUNTRY/10 CAR/10 MODEL/20 RCOST/6
MATCH *
START 80
STOP 200
DATA ON NEWTAPE
END
```

Prompting for Transaction Values

When the subcommand PROMPT is used then FOCUS will request that the operator type the data values in response to prompting messages. The operator uses the free format comma delimited method of describing data, but because only one field at a time is requested the operator need only type one field value then depress the carriage return. If several values are typed then they must be separated by commas. FOCUS will then skip ahead and resume prompting for any values not yet prompted.

Example: PROMPT entry of data

```
MODIFY FILE CARS
PROMPT COUNTRY CAR MODEL RCOST
MATCH COUNTRY CAR MODEL
ON MATCH UPDATE RCOST
ON NOMATCH REJECT
DATA
```

```
DATA FOR TRANS 1
```

```
COUNTRY    =   England
CAR        =   Austin
MODEL      =   Marina 2 door GT
RETAIL COST=   2549
```

```
DATA FOR TRANS 2
```

```
COUNTRY    =   England, Austin, Marina 4 door GT
RETAIL COST=   2499
```

```
DATA FOR TRANS 3
```

```
COUNTRY    =   END
```

When PROMPT is used it is necessary to name all of the fields of interest in the subcommands as these are the only ones prompted. However, if no MATCH subcommand is issued, then the assumption is match all.

The subcommand PROMPT * means to prompt for all fields in the file. They will be requested in the order they are described in the MASTER dictionary.

To end the PROMPT session the word END is typed in response to any prompt. If the word END is actually to be entered as data enclose it in quotes, e.g. 'END'.

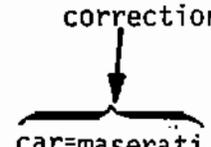
Correcting a Prior Response:

Until the last prompt for the transaction has been satisfied the operator has the opportunity to change the values supplied for any field. This is accomplished by replying to the current prompt, then typing a comma followed by fieldname=newvalue, where fieldname is the name of the field to be corrected, and newvalue is the replacement for the field.

Example: Correcting a prompted response

```
COUNTRY      = italy
CAR           = fiat
MODEL        = bora 2 door, car=maserati
RETAIL COST  = 31,000

DATA FOR....
```



Typing Ahead:

After several transactions are prompted the operator may become familiar with the sequence of prompts and want to supply several values at one time and so bypass the prompt message. This is accomplished by ending the current prompt with a comma, and supplying values for what would be the following prompts also between commas.

Example: Typing Ahead of the Prompts

```
MODIFY FILE CARS
PROMPT COUNTRY CAR MODEL MPG
MATCH COUNTRY CAR MODEL
ON MATCH UPDATE MPG
ON NOMATCH REJECT
COMPUTE CHANGEDATE
DATA
```

```
DATA FOR TRANS 1
```

```
COUNTRY = italy
```

```
CAR = fiat
```

```
MODEL = 128' 2 door
```

```
MPG = 23
```

```
DATA FOR TRANS 2
```

Note
typing ahead → COUNTRY = japan, datsun, B210 4 door, 29

```
DATA FOR TRANS 3
```

```
.  
.
.
```

Prompting for Groups of Fields:

Normally all of the fields named in the PROMPT subcommand will be requested for each transaction. However, one or more groups of fields can be repeated a fixed number of times. This number can be larger than needed because the operator can 'break out' of a group by typing an exclamation mark '!'. Prompting then resumes at the next higher group, or if no other group, from the start of the list of fields.

The syntax for specifying a group is to enclose the list of fields on the PROMPT subcommand in parenthesis with the replication number in front of the open parenthesis. e.g.

PROMPT field field n (field field field)

Example: Prompting with Groups of Fields

```
MODIFY FILE CARS
PROMPT COUNTRY CAR 5 (MODEL SEATS RCOST MPG)
MATCH COUNTRY CAR
ON NOMATCH REJECT
MATCH MODEL
ON NOMATCH INCLUDE
ON MATCH UPDATE SEATS RCOST MPG
DATA
```

```
DATA FOR TRANS 1

COUNTRY      = italy

CAR          = fiat

MODEL        = 128 2 door

SEATS        = 4

RETAIL COST  = 2741

MPG          = 22

DATA FOR TRANS 2

MODEL        = 128 4 door

SEATS        = 4

RETAIL COST  = 3049

MPG          = 21
```

continued

Example: Prompting with Groups of Fields (cont'd)

DATA FOR TRANS 3

MODEL = ! ← Note 'break out' of group

COUNTRY = japan

CAR = datsun

MODEL = B210 2 door

SEATS = 4

RETAIL COST = 2899

MPG = 27

DATA FOR TRANS 4

MODEL = B210 4 door, 5, 3184, 24 ← Note typing ahead

DATA FOR TRANS 5

.
.
.
.

Cancelling Prior Responses:

If the response to any prompt is the termination character, '\$', then the current transaction which is being assembled is cancelled. The following message is displayed:

(FOC309) TRANSACTION INCOMPLETE:

This allows the operator to cancel the current responses and start over again. The transaction number is incremented and the prompting starts from the beginning of the list of fields.

Annotating MODIFY Procedures

If an asterisk is the first non-blank character on a line in the MODIFY sub-command stream then the line is ignored. Hence, a procedure which is catalogued, for instance, can be annotated with comments which serve to document the procedure.

Example: Annotated procedure

```
MODIFY FILE CARS
*
*THIS PROCEDURE MATCHES EXISTING CARS
*AND UPDATES VARIOUS ITEMS OF SPECIFICATIONS
*
MATCH COUNTRY/10 CAR/10 MODEL/20
*
*NOTE ONLY EXISTING RECORDS ACCEPTED
*
ON NOMATCH REJECT
LOG NOMATCH ON BADIES
ON MATCH UPDATE LENGTH WEIGHT WBASE
ON MATCH UPDATE MPG SECONDS BHP
DATA ON NEWSPEC
END
```

Erasing a FOCUS File

A FOCUS file is a CMS file and can be erased by issuing the CMS ERASE command. For example, to erase a FOCUS file named PRODUCTS FOCUS on the A1 disk, the CMS command is:

```
ERASE PRODUCTS FOCUS A1
```

This can be issued from within FOCUS by typing:

```
CMS ERASE PRODUCTS FOCUS A1
```

File Status

Should a FOCUS file update run operating under the MODIFY command suddenly stop for any reason (computer goes down, telephone line is lost, etc.) a question may arise as to what was the last transaction to actually enter the data file. This information will be provided by issuing the FOCUS query command followed by the full name of the data file to be queried. The syntax is:

```
? FILE filename filetype filemode
```

For example:

```
? FILE CARS FOCUS A1
```

If the filemode is missing it defaults to A1. If the filetype is missing it defaults to FOCUS.

The display produced by this command appears as follows:

STATUS OF FOCUS FILE: CARS FOCUS A1 ON 01/18/76 AT 21.42.46					
SEGNAME	ACTIVE COUNT	DELETED COUNT	DATE OF LAST CHG	TIME OF LAST CHG	LAST TRANS NUMBER
ORIGIN	5	0	01/18/76	21.30.17	53
CARREC	10	0	01/18/76	21.30.17	53
MODREC	18	0	01/18/76	21.30.17	53
BODY	18	0	01/18/76	21.30.17	53
SPECS	18	0	01/18/76	21.30.17	53
WARANT	8	0	01/18/76	21.30.17	53
EQUIP	25	0	01/18/76	21.30.17	53
TOTAL	102	0			
FILE SIZE	31465				
LAST TRANSACTION			01/18/76	21.30.17	53

The column for TRANS NUM is the number of the last transaction on the data and time shown which entered the data base. This date and time match the one printed at the start of each MODIFY command. Change dates for each segment in the file are listed separately, and the last line is a file summary.

File Status (cont'd)

If the transaction file is available the run can be re-started by adding the START subcommand to the MODIFY procedure. For example, if the last transaction to modify the file was numbered 526, then the subcommand START 527 will skip over the first 526 'physical' transaction records. If a physical transaction is not equivalent to a 'logical' transaction then the former has to be adjusted by the number of physical records per logical record. For example, if 3 'cards' of 80 characters each form one logical record, then we wish to skip 3x526 or 1578 physical records and START 1579 would be used, as this positions us to the 527th logical record.

Subcommand	Summary of Subcommands Meaning and Usage
FILE	<p>Name of data file</p> <p>FILE filename</p> <p>Example:</p> <p>FILE CARS</p>
FORM	<p>Describe fixed format transaction records</p> <p>FORM fieldname/length fieldname/length...</p> <p>Example:</p> <p>FORM COUNTRY/10 CAR/16 MODEL/14 X5 SEATS/1</p>
MATCH	<p>Names of fields from transaction which must match corresponding data base fields</p> <p>MATCH fieldname fieldname fieldname...</p> <p>Example:</p> <p>MATCH COUNTRY CAR MODEL</p>
ON MATCH	<p>Direction of what happens when transaction fields match data base fields</p> <p>ON MATCH REJECT (no duplicates)</p> <p>ON MATCH UPDATE fieldname fieldname...</p> <p>ON MATCH DELETE</p> <p>ON MATCH COMPUTE name name...</p> <p>ON MATCH INCLUDE (allows duplicates)</p> <p>Example:</p> <p>MATCH COUNTRY CAR MODEL</p> <p>ON MATCH UPDATE SEATS LENGTH WEIGHT</p> <p>ON MATCH UPDATE HEIGHT DCOST RCOST</p>
ON NOMATCH	<p>Direction of what happens when transaction fields do not match data base fields</p> <p>ON NOMATCH REJECT (transaction did not find corresponding values)</p> <p>ON NOMATCH INCLUDE (add new record to data base)</p>

Subcommand	Summary of Subcommands (cont'd) Meaning and Usage
VALIDATE	<p><u>Example:</u></p> <pre>MATCH COUNTRY first match COUNTRY ON NOMATCH REJECT if no match reject transaction MATCH CAR MODEL if match on COUNTRY then contin- ON NOMATCH INCLUDE ue to match CAR and MODEL if no match add a new record</pre> <p>List of pre-defined test conditions which must be true (have a non-zero value) in order for a transaction to be considered valid.</p> <p>VALIDATE fieldname fieldname...</p> <p><u>Example:</u></p> <pre>DEFINE FILE CARS SEAT-TEST=IF SEATS GE 1 AND SEATS LE 8 THEN 1 ELSE 0 ; WEIGHT-LIMIT=IF WEIGHT LE 5000 THEN 1 ELSE 0 ; POWER-TEST=POWER GT 0 AND RPM GT 0 OR POWER EQ 0 AND RPM EQ 0 ; END MODIFY FILE CARS VALIDATE SEAT-TEST WEIGHT-LIMIT POWER-TEST . . etc.</pre>
COMPUTE	<p>List of calculations to be performed on transaction data fields.</p> <p>COMPUTE WEIGHT 'RETAIL COST'</p>

Subcommand	Summary of Subcommands (cont'd) Meaning and Usage
START	<p><u>Example:</u></p> <pre> DEFINE FILE CARS WEIGHT=IF COUNTRY EQ 'ITALY' THEN WEIGHT/2.2 ELSE WEIGHT ; RETAIL COST=IF RCOST EQ 0 THEN DCOST*1.30 END MODIFY FILE CARS FORM COUNTRY CAR MODEL DCOST WEIGHT/6 RCOST/6 MATCH COUNTRY CAR MODEL ON MATCH UPDATE WEIGHT RCOST ON NOMATCH REJECT COMPUTE WEIGHT 'RETAIL COST' DATA ON FORCARS </pre> <p>Number of the transaction at which processing is to start. Default is 1.</p> <p>START nnnn</p> <p><u>Example:</u></p> <p>START 1001</p>
STOP	<p>Number of the transactions at which processing is to stop. Default is all transactions.</p> <p>STOP 2000</p>
*	<p>An asterisk as the first non-blank character tags the line as a comment and the line is ignored. This is useful for annotating procedures with comments.</p>

Subcommand	Summary of Subcommands (cont'd) Meaning and Usage
FREEFORM	<p>Specify the order of submission of comma delimited free format data elements.</p> <p>FREEFORM fieldname fieldname fieldname</p> <p><u>Example:</u></p> <p>FREEFORM CAR MODEL LEN WEIGHT</p>

File Maintenance Error Messages

- (FOC401) SUBCOMMAND IS NOT RECOGNIZED:
The subcommand issued in a MODIFY procedure is not recognized. Check the list of subcommands.
- (FOC402) UNRECOGNIZED WORD:
The syntax of the subcommand does not recognize a word. Check the syntax of the subcommand.
- (FOC403) TRANSACTION DATA FILE CANNOT BE LOCATED:
The name of the file containing the transactions is provided on the DATA ON ddname subcommand. A FILEDEF may be missing for this name so it can't be located.
- (FOC404) INCOMPLETE PROCEDURE:
The MODIFY procedure was not executed due to an error in the procedure, or termination by a QUIT or END by the operator before the end of the procedure. The message is a confirmation of the status of the procedure.
- (FOC405) TRANS n REJECTED...DUPLICATE
The MATCH conditions for transaction number n all match corresponding data base records. The option ON MATCH REJECT is in effect (explicitly or by default) and the transaction is rejected.
- (FOC406) FIELDNAME IS NOT RECOGNIZED:
The name of the data field is not on the list of fields for this file. Check the spelling, or the list of fieldnames and aliases for the file.
- (FOC407) TRUNCATED FIELDNAME IS NOT UNIQUE:
The short form used to identify a data field is not unique and could also apply to another field in the file. Use additional letters, or the full name to make it unique.
- (FOC408) INCORRECT OPTION AFTER 'ON':
The actions after the word 'ON' are only ON MATCH, or ON NOMATCH.
- (FOC409) CONFLICTING SUBCOMMANDS:
The subcommand conflicts with a previous one. They are mutually exclusive, i.e. when one or the other. For instance, ON MATCH DELETE conflicts with ON MATCH UPDATE.

File Maintenance Error Messages (cont'd)

- (FOC410) ACTIVITY AFTER WORD MATCH IS NOT RECOGNIZED:
The action used after MATCH is either UPDATE, DELETE, REJECT, or EXEC. Anything else is not recognized.
- (FOC411) ACITVITY AFTER WORD NOMATCH IS NOT RECOGNIZED:
The action word after NOMATCH is either INCLUDE, or REJECT. Anything else is not recognized.
- (FOC412) FIELDNAME IS NOT A DEFINED VALIDITY TEST:
The name used in a VALIDATE subcommand does not have a DEFINE'd expression. Issue a ? DEFINE command for the current list of DEFINE'd names.
- (FOC413) NULL TRANSACTION AND/OR DUPLICATE TRANSACTION:
Two transactions are the same and the option ON MATCH REJECT is in effect explicitly or by default, or insufficient data appears on the transaction and the procedure hasn't given an action for its processing.
- (FOC414) NULL INCLUDE:
The transaction does not contain data for descendent segments although the procedure explicitly required them.
- (FOC415) TRANS n REJECTED...NOMATCH:
Transaction number n does not match corresponding data base fields as specified by the MATCH subcommand. The name of the segment (SEGNAME) on which no match occurred is displayed for reference, followed by the full transaction record.
- (FOC416) THE NUMBER OF FIELDS EXCEEDS THE NUMBER IN THE DICTIONARY:
Freeform data in which comma's delimit each field contain too many comma's. Or, a field is identified by its order number in the dictionary which exceeds the number of fields in the file.
- (FOC417) ADDITIONAL CORE IS NEEDED FOR RECORD WORK SPACE:
Before processing of transactions start work space is obtained. If this is not adequate then more core is needed. Add 12K to the amount of current storage and issue the CMS command DEFINE STORAGE AS N.

File Maintenance Error Messages (cont'd)

(FOC418)

(FOC419) FORM SUBCOMMAND ELEMENT or FIELDNAME NOT RECOGNIZED:

An element on the FORM subcommand such as the number of characters to process, cannot be identified.

(FOC420) FIELD NAMED DOES NOT HAVE A COMPUTED EXPRESSION:

The name used on a COMPUTE subcommand does not appear in the list of computed items. Issue the ? DEFINE command to obtain the current list.

(FOC421) TRANS n REJECTED...INVALID:

Transaction number n failed the VALIDATE test for it. The name of the test is displayed, followed by the complete transaction.

(FOC422) FOCUS FILE IS NOT ON WRITABLE DISK, CAN'T MODIFY:

The MODIFY command can only be issued for FOCUS files which are on writable disks.

(FOC423) ERROR WRITING LOG FILE NAMED:

An input/output error has occurred while attempting to write a transaction to the log file named. Possible error is an incorrect FILEDEF. Check the length parameter, LRECL, it must be large enough to permit a full 'logical' transaction to be written.

(FOC424) TYPE OF LOG RECORD NOT RECOGNIZED:

The activity to LOG is not on the allowed list. e.g. NOMATCH, INVALID, DUPLS, ERROR, TRANS, or ACCEPTS.

(FOC425) SYNTAX ERROR ON LOG SUBCOMMAND:

A word is not recognized in the LOG SUBCOMMAND.

(FOC426) LOG FILE MUST HAVE FILEDEF...PLEASE ISSUE IT FOR:

A FILEDEF for the particular LOG file has not been issued prior to the MODIFY procedure. LOG files must have FILEDEF's.

(FOC427) LOG MESSAGES ARE EITHER ON OR OFF:

The syntax in the LOG subcommand mentions messages but is not either MSG ON or MSG OFF.

File Maintenance Error Messages (cont'd)

(FOC428) DATA VALUE HAS INCORRECT FORMAT (EX:NOT NUMERIC):

On a free form transaction (comma delimited) a field value either exceeds the length specified for it, or a numeric field is given a non-numerical character. These are both format errors since they violate the FORMAT element for the field as stored in the MASTER dictionary.

(FOC429) NESTED GROUPS ON FORM SUBCOMMAND, USE UNNESTED ONLY:

The FORM subcommand specifies repeating groups of fields which are enclosed between parenthesis. Within one set of parentheses another set has been detected, hence nesting the groups. This is not permitted.

(FOC430) ATTEMPT TO INCLUDE ANOTHER SEGMENT IN UNIQUE CHAIN:

A segment has a SEGTYPE=U (unique). An attempt to include more than one segment on this chain has been detected. The transaction is treated as a duplicate and rejected. It is logged on the DUPLS LOG file if one has been specified. The duplicate acceptance option, ON MATCH INCLUDE, does not apply to unique segments.

(FOC431)

(FOC432) CANNOT MATCH ON, OR UPDATE DEFINED FIELD:

The list of fields to match on the MATCH subcommand must all exist in the data file, they cannot be temporary defined fields unless these have the same name as real fields.

(FOC433) START/STOP SUBCOMMAND REQUIRES INTEGER NUMBER:

The value following a START or STOP subcommand is not a positive integer number.

(FOC434) PROCEDURE CONTAINS CONFLICTING SUBCOMMANDS:

The MODIFY procedure contains instructions which are mutually exclusive, much as FIXFORM, and also FREEFORM.

)

File Maintenance Error Messages (cont'd)

(FOC435) MATCH CONDITIONS MISSING...SUPPLY THEM FIRST:

An ON MATCH, or ON NOMATCH subcommand precedes any MATCH condition. A line may have been left out of the procedure or inserted at the wrong point.

(FOC436) SOURCE OF DATA MISSING...NO DATA COMMAND:

The subcommand DATA or DATA ON ddname must be present in the procedure to identify the source of the transactions or start of data entry.

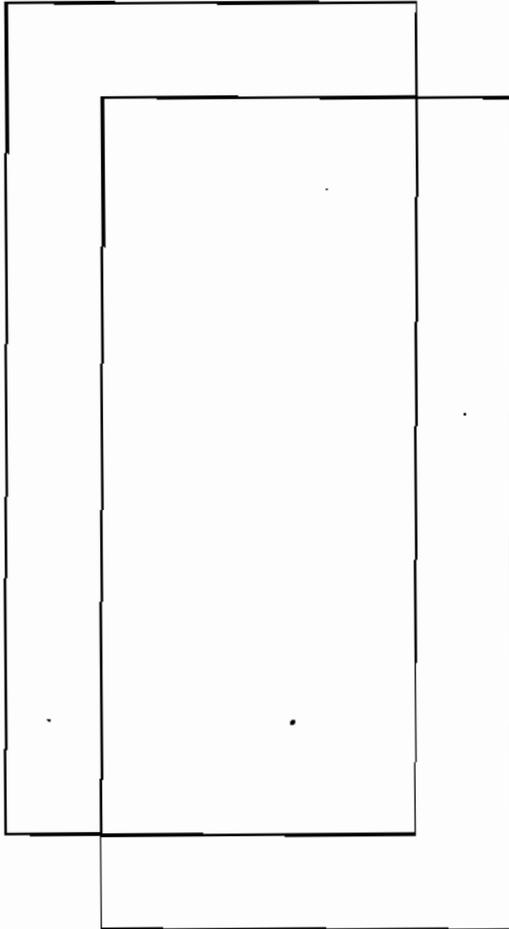
(FOC437) FORMAT ON FIXFORM INCOMPATIBLE WITH DATA FORMAT:

The type and/or length specifications on a FIXFORM element is not compatible with its FORmat attribute. i.e. COST/P4 and COST is described as D8.2.

) (FOC438) CANNOT MODIFY A FILE WHOSE FILESUFFIX IS NOT 'FOC'..?

Only MASTER file descriptions containing the attribute SUFFIX=FOC are considered FOCUS files. Add this value to the file description.

)



F O C U S

USER'S MANUAL

INFORMATION BUILDERS INC.
254 WEST 31st STREET
NEW YORK, N.Y. 10001
(212) 736-4433

TABLE OF CONTENTS

SCAN CONCEPTS5-01
FOCUS SCANNER5-01
Modes of Operation - SCAN/INPUT5-01
Using the Scanner5-02
Identifying Data Fields5-02
Print Suppression5-03
Subcommand Truncation5-03
Current Position Concept5-04
Showing Fields of Interest5-05
SCAN Display Line5-07
SCAN SUBCOMMANDS5-08
Summary of SCAN Subcommands5-08
AGAIN5-10
BACK5-11
CHANGE5-12
CMS5-14
DELETE5-15
DISPLAY5-17
FILE5-18
INPUT5-19
JUMP5-20
LOCATE5-21
MARK5-22
MOVE5-23
NEXT5-24
?5-25
QUIT5-26
REPLACE5-27
SAVE5-29
SHOW5-30
TOP5-32
TYPE5-33
UP5-34
X and Y5-35

FOCUS SCANNER

The FOCUS SCAN command allows a user to interactively edit a FOCUS file by using subcommands very similar to those used by the CMS EDITOR to EDIT CMS files.

The SCAN command is a FOCUS facility which allows a user to:

- . Add records to a new, or existing FOCUS file.
- . Delete records from a FOCUS file.
- . Change the data values of fields in a FOCUS file.
- . Search a FOCUS file based on search criteria.
- . Display the contents of records showing all of the field values, or a subset of the fields in a FOCUS file.
- . Move (Relink) record segments and its descendents from one parent record to another parent record when the FOCUS file has a parent-descendent hierarchical structure.

Modes of Operation

SCAN/INPUT

The Scanner has two modes of operation, a SCAN mode, and an INPUT mode. When in SCAN mode all of the Scanner subcommands can be issued to search, display, and change the contents of a FOCUS file. By typing the subcommand INPUT the INPUT mode is entered. All lines typed from this point are input lines and add new records, or record segments, to a FOCUS file. The comma delimited free format method of entering records is used for this purpose when in INPUT mode. (See Section 3 FOCUS User's Manual, Describing Free Format Records). When a null line of at least one character followed by a carriage return is typed the Scanner switches back to SCAN mode.

The SCAN mode is entered from within FOCUS by typing the word SCAN followed by the word FILE then the name of the FOCUS file.

Using the Scanner

The SCAN mode is entered in one of two ways. It can be entered from within FOCUS by typing the word SCAN followed by the word FILE then the name of the FOCUS file.

```
SCAN FILE filename
```

For instance:

```
SCAN FILE CARS
```

Identifying Data Fields

A natural difference between a CMS file and a FOCUS file is the use of fieldnames to identify items of information. A number of SCAN subcommands require that particular data fields be referenced. For instance to locate a record in which a data field or fields have a given value. i.e.

```
LOCATE CAR=FIAT, MODEL=128 2 DOOR AUTO
```

Data fields may be identified in four ways:

- . By their full fieldname (the one stored in the MASTER FOCUS dictionary).
- . By their full alias, or alternate name (also stored in the MASTER FOCUS dictionary).
- . By the shortest unique truncation for the fieldname or alias.
- . By the relative position number of the field as it is described in the MASTER FOCUS dictionary.

Example: Identifying Fields

A file contains the fields of

<u>Fieldname</u>	<u>Alias</u>	<u>Typed name</u>	<u>Refers to</u>
COUNTRY	CO	MODEL	MODEL
CAR-NAME	CAR	MOD	MODEL
MODEL	MOD	M	MODEL
BODY-TYPE	TYPE	CA	CAR-NAME
SEATS	S	C	errors, not unique
RETAIL-COST	RCOST	R	RETAIL-COST

Print Suppression

After subcommands which locate or affect records are processed the record affected is displayed. For instance after LOCATE, NEXT, JUMP, CHANGE, and UP. The printing of the record may be suppressed if a period is typed immediately after the subcommand. Hence,

NEXT	will retrieve and display the next record
NEXT.	will retrieve but not display the next record.

This is convenient when 'global' operations which affect many records are being performed.

Subcommand Truncation

A subcommand or its shortest unique truncation may be typed. In the Summary of Subcommands the letters which are capitalized are the shortest unique truncation for that subcommand.

Current Position Concept

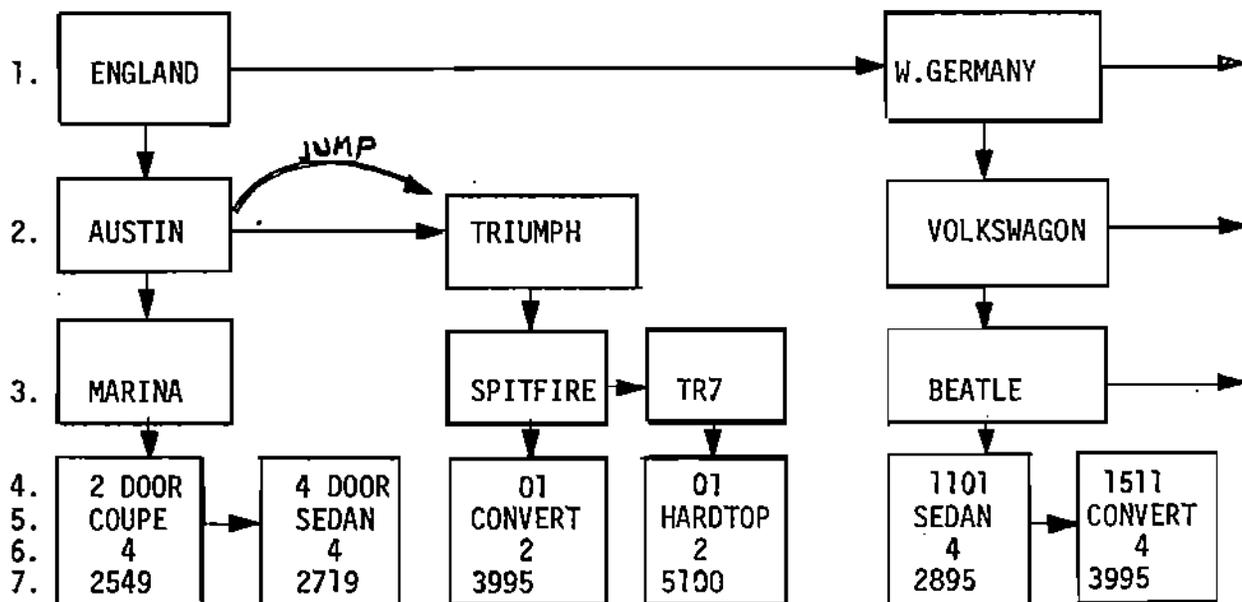
A FOCUS file is not a sequential file with one data record following another like a CMS file. Hence, the notion of current line pointer from CMS files is not applicable for FOCUS files. However, a somewhat similar effect is achieved by viewing the hierarchical aspect of the FOCUS file in a 'top down, left-to-right' scanning direction. Consider a file with 4 record segments structured as:

COUNTRY

CAR

MODEL-TYPE
 BODY-TYPE
 SEATS
 RETAIL-COST

Data for the file appears schematically as:



There are 7 data fields in the file organized on 4 levels of hierarchy.

If the current position is:

```
ENGLAND AUSTIN MARINA 2 DOOR COUPE 4 2549
```

then the subcommand NEXT will move to:

```
ENGLAND AUSTIN MARINA 4 DOOR SEDAN 4 2719
```

A series of NEXT subcommands will move through each record and thereby scan the entire file. If, however, the subcommand was JUMP CAR then the next displayed record would be:

```
ENGLAND TRIUMPH SPITFIRE 01 CONVERT 2 3995
```

A series of JUMP subcommands will then move very rapidly through a file by using the structure to find the next occurrence of a given field.

Showing Fields of Interest

The number of data fields in a FOCUS file and their corresponding display lengths will in many cases require several lines to print. A SCAN subcommand named SHOW is provided to allow a user to select a subset of fields. Only the named fields will then be displayed. For instance if the subcommand is:

```
SHOW COUNTRY CAR MODEL
```

Then only these three fields will be shown.

If the current position is

```
ENGLAND AUSTIN MARINA
```

and the subcommand NEXT is issued the values displayed next are:

```
ENGLAND TRIUMPH SPITFIRE
```

This is the next occurrence of the fields desired, not necessarily the physically next record in the file.

Note an effect of the SHOW subcommand is to consider the file as consisting of only the fields named. Since the three fields in the example do not include any on the 4th level of the file this level is not retrieved.

When a file structure consists of multiple top-to-bottom paths (See Section 3 Users Manual Describing FOCUS Files) then the use of the SHOW subcommand is required to select one of the paths for display.

SCAN Display Line

When a record is displayed in the SCAN mode each data field is identified by the alternate (alias) name which the user has assigned to the field. The alias name is one of the file description elements which is stored in the MASTER dictionary at the time the file is created. This name should be a short unique abbreviation as its main purpose is to serve as a quick identifier for a field so as to reduce typing, and attendant typing errors.

Example:

If the fieldnames and alias' are:

<u>Fieldname</u>	<u>Alias</u>
COUNTRY	CTY
CAR NAME	CAR
WEIGHT	WGT
RETAIL COST	RCOST

then the SCAN subcommand appears as:

TYPE 2

CTY=ENGLAND CAR=AUSTIN WGT=4724 RCOST=3896

CTY=ENGLAND CAR=TRIUMPH WGT=3124 RCOST=6124

Summary of SCAN Subcommands

<u>Subcommand</u>	<u>Parameters</u>	<u>Function</u>
Again		Repeat the last subcommand.
Back		Go back to a prior Mark.
CHANGE	field= /string/string/n	Change a string in n occurrences.
CMS		Issue CMS commands from SCAN.
DElete	field n	Delete n occurrences of the field.
Display	field field	Prints values for fields shown.
File		End the session, save all results.
Input		Enter Input mode.
Jump	field n	Jump to nth occurrence of field.
Locate	field=value	Search for records which match.
Mark		Mark a record.
MOve	fieldname	Relink to another parent.
Next	n	Move n records ahead.
?		Print last subcommand.
Quit		End the session, kill pending changes if possible.
Replace	field=value	Replace a field value.
Save		Save all results, continue.
SHow	field field	Display list of fields.
TOp		Reposition to 'top' of file.
Type		Type n records.

Summary of SCAN Subcommands (cont'd)

<u>Subcommand</u>	<u>Parameters</u>	<u>Function</u>
UP		Up to first record of parent.
X		Assign a SCAN subcommand, or execute one.
Y		Assign a SCAN subcommand, or execute one.

AGAIN

Again	:
-------	---

The last valid subcommand typed prior to this one will be repeated.

The subcommand is particularly useful when typed after a LOCATE as it will continue the search for another record with the same test conditions.

Example:

LOC SEATS=5

AGAIN

The first record starting from the current position which matches the test condition is presented. The process is repeated as if the LOCATE had been retyped, and the second record meeting the test conditions will be displayed.

BACK

BAck	
------	--

The BACK subcommand is issued to return to a record which has been previously noted via the MARK subcommand.

Example:

BACK

Example:

```
SHOW COUNTRY CAR
LOC CAR=MAZDA
COUNTRY=JAPAN CAR=MAZDA
MARK
NEXT
COUNTRY=JAPAN CAR=DATSUN
BACK
COUNTRY=JAPAN CAR=MAZDA
```

CHANGE

Change	field=/old string/new string/,\$	1 * n
--------	----------------------------------	-------------

The Change subcommand will locate a string of characters within a data field and substitute another string for them.

The first character typed after the equal (=) sign is used as the string delineator. It must then be typed at the end of the string to be changed, and at the end of the new string.

The replication factor default is 1. If the process of changes is to affect more than one record then a line terminator of a comma followed by a dollar sign is typed, before the replication factor.

Example: TYPE
 CAR=TOYOTA MODEL=COROLA
 CHANGE MODEL=/OLA/OLLA/
 CAR=TOYOTA MODEL=COROLLA

A period after the subcommand suppresses the display of the changed record. This is useful when a global change is performed. Changes may be made sequentially, or to all records which match a LOCATE criteria.

Sequential Changes:

The current position is reached without a LOCATE subcommand. All occurrences of the old string are changed to the new string throughout the data base.

Example: TOP
 NEXT
 CHANGE. BODY=/COUP/COUPE/ ,*\$

Matched Changes

The current position is reached because the immediately prior subcommand was LOCATE. The test conditions of this LOCATE are retained and the changes performed only on retrieved records meeting the LOCATE condition.

Example: TOP
LOCATE. COUNTRY=JAPAN
CHANGE. BODY=/COUP/COUPE/ , \$*

.CMS

CMS	
-----	--

The SCAN subcommand CMS is followed on the same line by any valid CMS command.

The line is passed to CMS for processing and any messages which result are CMS messages.

Example:

```
CMS LISTF * FOCUS ( LABEL
```

The CMS command LISTF is issued.

DELETE

DElete	fieldname	{ * n
--------	-----------	-------------

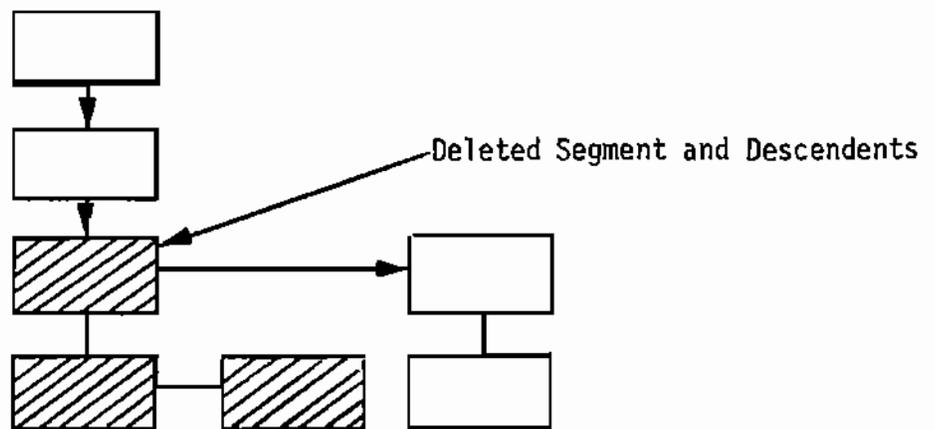
The segment containing the fieldname is deleted and all descendent segments to it are deleted. Any references to indexed fields are removed from their associated indexes.

Example:

```

SHOW COUNTRY CAR MODEL
LOC CAR=DATSUN
DELETE MODEL
SEGMENTS DELETED=3
    
```

The number of record segments deleted is displayed after the subcommand has executed.



Deletion of Selected Records

If the immediately prior subcommand is LOCATE, then the record selection criteria is preserved and should the DELETE subcommand have a repetition factor greater than one, then only records meeting the select criteria are deleted. If no record select criteria are in effect

at the time of deletion then sequential segments are removed.

Example: Selected Deletions

SHOW COUNTRY CAR MODEL

LOC COUNTRY=ITALY

DEL CAR *

SEGMENTS DELETED=30

All car segments in COUNTRY=ITALY are removed.

DISPLAY

Display	fieldname fieldname fieldname
---------	-------------------------------

The DISPLAY subcommand types to the terminal the current values of the fields whose names are provided in the list. The list of fieldnames are separated from each other by a blank space. The field identifier may be the full fieldname, alternate alias, or shortest unique truncation. Or, the list may be replaced by an asterisk '*', meaning to display all fields.

The DISPLAY subcommand is used in situations where a SHOW subcommand is in effect and not all of the data values are present in the currently shown print line. It is convenient for example to move through a file with a minimum set of field values being shown. Then at desired positions expand the list of displayed value, just for that position.

The DISPLAY subcommand does not stay in effect. It causes a list of field values to be displayed only at the instance of issuance. If, it is to be issued repetitively then it should be stored in the 'X' subcommand.

Example:

```
DISP      SEATS  LEN  WEIGHT
```

The DISPLAY subcommand does not cause a retrieval to occur, hence, the list of fields named must be within the current span of fields. If a field named is outside the span of the currently retrieved segments then a dot '.' is displayed in place of a value for the field.

FILE

File	
------	--

The SCAN session is ended and all modifications are permanently written to the FOCUS file.

Example:

FILE

The session is ended.

Note: The word END issued as a subcommand is a synonym for FILE and results in a normal ending of the session.

INPUT



The subcommand Input changes the SCAN mode from SCAN to INPUT. All lines typed from this point forms new record segments. A null line consisting of at least 1 blank character followed by a carriage return causes the INPUT mode to be left and the SCAN mode re-entered.

The input records use the comma delimited free format method of description (See Section 4 Describing Free Format Transactions).

The new records added are inserted after the record displayed in the current position, that is, they break the 'chain'. However, if the segment is being maintained in sort sequence, then a check is performed and the new records inserted in their proper position.

Example:

```

select
display → SHOW COUNTRY CAR MODEL TYPE SEATS
Locate
record → LOC COUNTRY=JAPAN, CAR=DATSUN
record
retrieved → CTY=JAPAN CAR=DATSUN MODEL=B210 2 DOOR TYPE=SEDAN SEATS=4
enter
input → INPUT
type new
records → MODEL=610 2 DOOR, TYPE=HARDTOP, SEATS=4, $
          MODEL=260Z 2 DOOR, TYPE=COUPE, SEATS=2, $
blank line _____
re-enter
scan → SCAN:

```

JUMP

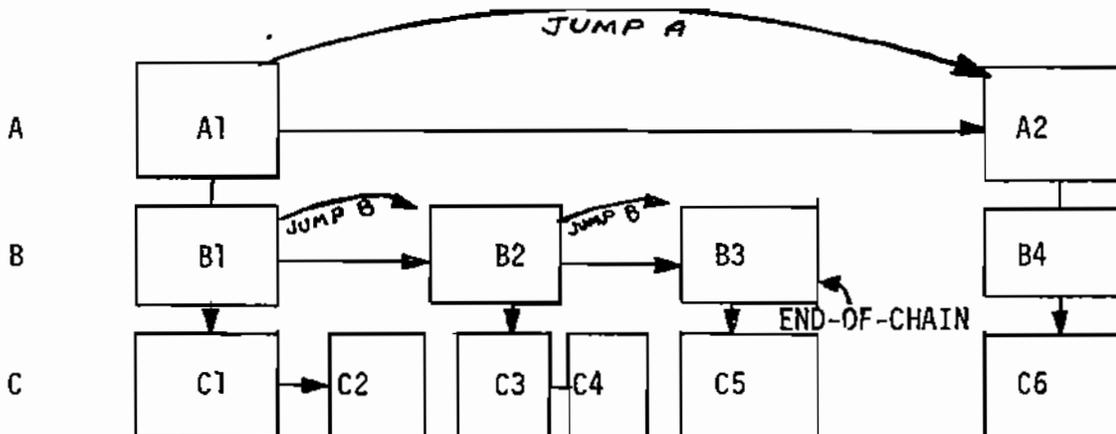
Jump	fieldname	1
		*
		n

Starting at the fieldname in the record at the current position an immediate move to the nth occurrence of the same field is made. This passes over any descendent records and is a rapid way to traverse a FOCUS file.

Should no more occurrences of the field be encountered for the same 'parent' record then the jump does not continue to the start of the next chain but stays at the last record in the current chain and the message END-OF-CHAIN is displayed.

For example, in the diagram if the current position is A1 B1 C1, Then;

JUMP B moves from A1 B1 C1 to A1 B2 C3
 JUMP B 2 moves from A1 B1 C1 to A1 B3 C5
 JUMP B 3 moves from A1 B1 C1 to END-OF-CHAIN



In order to proceed from
 A1 B3 C5 to A2 B4 C6
 the command NEXT is issued, or JUMP A.

LOCATE

Locate	field=value, field=value, \$	1 * n
--------	------------------------------	-------------

Starting from the current position a search for the records having the values requested is started. When a match is found the record is displayed. This is attempted n times. The default repetition factor is 1. If the repetition factor is set '*' then all records meeting the match conditions are displayed. If the end of the file is encountered in the search then EOF: will be displayed.

The terminator signal of a '\$' is not required if only one record is sought, but is needed if a repetition factor is to be provided.

Example:

LOC COUNTRY=JAPAN, CAR=MAZDA, \$*

All records starting from the current position which match the test condition are displayed.

MARK

Mark	
------	--

The MARK subcommand is issued to preserve the locational information about a record so that it can later be returned to by the issuance of a BACK subcommand. Also, it identifies a record which is to be moved via the MOVE subcommand to a new location in the file.

Example:

MARK

Example:

```
SHOW COUNTRY CAR
LOC CAR=FIAT
MARK
LOC CAR=DATSUN
COUNTRY=JAPAN CAR=DATSUN
BACK
COUNTRY=ITALY CAR=FIAT
```

MOVE

MOve	fieldname
------	-----------

The MOVE subcommand is issued to move a record segment (and all its descendent segments) from one 'parent' to another parent segment.

The record to be moved is noted by a MARK subcommand. A new location in the file for the marked record is reached in any manner, (LOCATE, NEXT, etc.), and the MOVE subcommand issued followed by the fieldname identifying the segment to be moved. The segment moved will become a descendent of the parent at the current position. If there are other descendent records then it will be located in the proper sort sequence when the SEGTYPE is S or SH (sorted, sorted high-to-low). If the SEGTYPE is not S or SH then it will be inserted after the record currently shown.

Example:

<p>Italy</p> <p>Datsun Fiat</p> <p>B210 128 4 Door Coupe Sedan</p>	<p>Japan</p> <p>Toyota Datsun</p> <p>Corolla 2 B210 door Coupe Sedan</p>
---	--

```

SHOW COUNTRY CAR BODY
LOC CAR=DATSUN
COUNTRY=ITALY CAR=DATSUN BODY=COUPE
MARK
JUMP COUNTRY
COUNTRY=JAPAN CAR=TOYOTA BODY=SEDAN
MOVE CAR
COUNTRY=JAPAN CAR=DATSUN BODY=COUPE
    
```

NEXT

Next	1 * n
------	-----------------

The current position is advanced n records and the result displayed. If the end of the file has been encountered during the span of the movement of the current position then the reply EOF: will be displayed.

The default is 1 record.

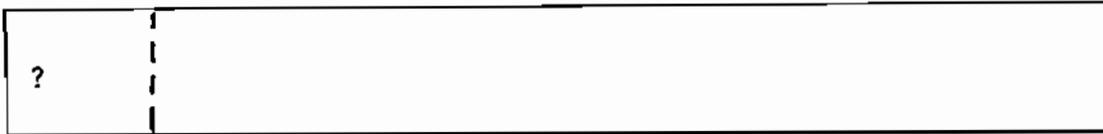
A period after the subcommand will suppress the display of the record.

Example:

NEXT 4

The 4th record from the current position is retrieved and displayed.

?



The ? subcommand will recall and display the last recognized subcommand issued in the SCAN mode.

Example:

```

TOP
LOC  CAR=DATSUN  ← sub-command
?
LOC  CAR=DATSUN  ← request for recall
                                ← display of last subcommand
    
```

QUIT

Quit	
------	--

The SCAN session is ended. All of the modifications to the file which have not yet been written permanently to the disk will be suppressed.

The use of this subcommand does not guarantee that all changes to the file will be ignored. During SCAN execution there are large 'buffer' areas storing data file records. The QUIT subcommand acts only on these 'buffer' areas to prevent their transference to the disk.

If immediately prior to issuing this subcommand a change to the file was made, then it is highly probable that it can be suppressed. The more SCAN activity between a modification to the file, and the issuance of a QUIT the less likely the chance of suppression as the 'buffer' work areas may of necessity had to been written to the disk to make way for more pages of data file records.

Example:

QUIT

The session is ended.

REPLACE :

Replace	field=value, field=value, \$	1 * n
---------	------------------------------	-------------

The data values provided will replace those in the file for the record at the current position.

One, all or n field values of the given fields can be replaced at one time.

The fields replaced may reside on the same segment or different segments.

If the replication factor is greater than 1 then all of the replaced fields must reside on the same segment.

Two types of global replace operations can be specified.

. Sequential replacement

The current position was not reached via an immediately prior LOCATE subcommand. The replication factor then applies to this record and the next n-1 records retrieved.

. Matched replacement

The immediately prior subcommand was LOCATE. This established search criteria for retrieving the record at the current position. These search criteria remain in effect and the replication factor then applies to this and the next n-1 records which also meet the search criteria.

Example: Zero out all values for two fields

```
TYPE
REPLACE  AMOUNT=0, RCOST=0, $ *
```

Example: Set to 1 all values of certain records

```
LOCATE  MONTH=FEB, CODE=A
REPLACE. AMOUNT=1, $ *
```

The print suppression period after the REPLACE subcommand is particularly useful when multiple records are changed.

SAVE

Save	
------	--

All modifications to the FOCUS file are written out. The session continues in SCAN mode.

Use of this subcommand periodically during a SCAN session is recommended. Should the telephone line be lost or other processing interruption occur then the modifications to the file will not have to be repeated, at least to the last issuance of SAVE.

Example:

SAVE

All modifications to the file are written to the disk if not already written.

SHOW

SHOW	fieldname	fieldname	fieldname
------	-----------	-----------	-----------

The SHOW subcommand selects the data fields which the user wants to display. A list of fieldnames is provided with each name separated by a blank from the next. The field identifier may be the full fieldname, alternate alias, or shortest unique truncation.

If the SHOW subcommand is issued without a list of fieldnames then the names of the fields currently being shown will be displayed for user reference.

Upon entry to the SCAN environment all of the data fields in the first physical top-to-bottom path will by default be shown.

Example:

SHOW COUNTRY CAR MODEL SEATS

The print line will display the data from the four fields selected.

Displaying All Fields:

If an asterisk is used in place of a list of field names then it means to 'show all' fields.

Example:

SHOW *

SHOW (cont'd)

Displaying a Sub-Set of Fields:

An asterisk between two field names means to show all fields between the ones named including those named.

Example:

```
SHOW CAR * MPG
```

All fields from CAR up to and including MPG will be shown.

Example:

```
SHOW * MODEL
```

All fields up to and including MODEL will be shown.

Example:

```
SHOW CAR*
```

All fields starting from CAR will be displayed.

Example:

```
SHOW COUNTRY * MODEL LENGTH * MPG
```

Inquiring About the Current List of Shown Fields:

When the BRIEF subcommand is in effect then only the data values without the field identifier are displayed. The names of the fields can be requested by typing SHOW ?. The names are aligned over the data columns in the fashion of column titles for easier reference.

.TOP

Top	
-----	--

The current position is set to just before the first record. If the next subcommand is TYPE the first record will be retrieved and displayed.

After the message EOF: has been printed after any subcommand it is necessary to reset the current position. The TOP subcommand performs this function.

Example:

TOP

The current position is re-positioned to the top of the file, in front of the first record.

TYPE n

Type		1
		*
		n

The record at the current position is displayed plus the next n-1 records if the replication factor is greater than 1.

The default replication factor is 1.

If Type * is issued then after the last record is displayed the message EOF: is printed and the current position remains at end of file (EOF) until a TOP subcommand repositions the current position.

Example:

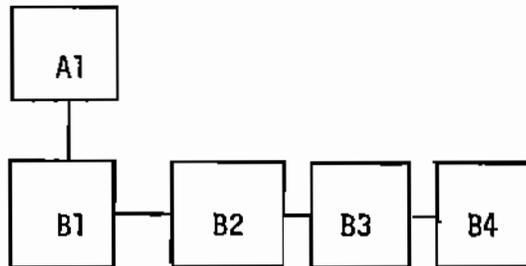
TYPE 5

The record at the current position and the next 4 are displayed.

UP

UP	fieldname
----	-----------

The UP subcommand will re-set current the current position to the first descendent under the given fieldname. Hence, it moves the position to the start of the current chain. For instance, if the current position is A1-B3, then UP A will re-set the current position to A1-B1.



Example:

```
LOC CAR=DATSUN, BODY=COUPE
UP CAR
```

The current position is re-set to the first MODEL under the CAR=DATSUN.

X and Y

X or Y	Subcommand
--------	------------

The X and Y subcommands are used to store a complete SCAN subcommand for later execution by the simple typing of the letter X or Y.

To set but not execute a value for X, or Y, it is typed as the first letter in front of any other subcommand. For instance,

```
X LOC MONTH=4, CODE=B
```

To execute X or Y it is typed alone. For instance,

```
X
```

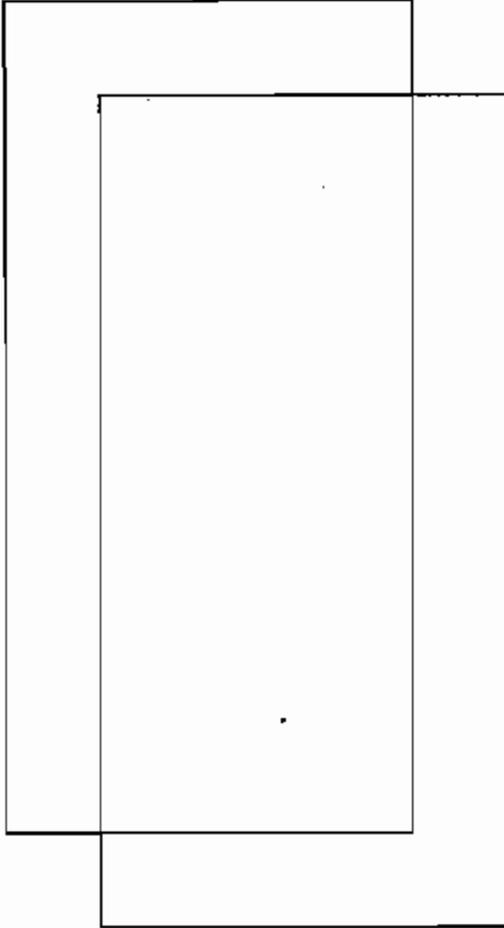
The print suppression control, and the replication factor is taken from the stored subcommand.

A series of operations can be performed by repeated X and Y subcommands.

Example:

```
X LOC CAR=COUPE
Y DELETE CAR
X
Y
X
Y
```

The LOC and DELETE are executed two times.



FOCUS USER'S MANUAL

INFORMATION BUILDERS INC.
254 WEST 31st STREET
NEW YORK, N.Y. 10001
(212) 736-4433

TABLE OF CONTENTS

INTRODUCTION6-01
Functions Performed by the HLI6-01
Program and HLI6-02
Constructing a Load Module6-02
Core layout during HLI execution.6-03
CONVENTIONS6-04
Communicating with the HLI.6-04
Syntax of CALL to Subroutine FOCUS6-05
Summary of HLI Commands.6-06
Syntax and Arguments on HLI Commands6-07
Record Work Area6-08
COMMANDS6-10
HLI Command OPN (Open)6-10
HLI Command SHO (Show)6-11
Relation of SHO List to:6-12
Test Relation List6-12
Change Field Identity List.6-12
HLI Command NEX (Next)6-13
HLI Command NXP (Next Physical)6-15
HLI Command NXD (Next thru index)6-17
HLI Command CLO (Close).6-19
HLI Command CHA (Change)6-20
HLI Command INP (Input).6-22
HLI Command SAV (Save)6-23
HLI Command DEL (Delete)6-24
HLI Command FST (First).6-25
HLI Command INFO (Information)6-27
CONCEPTS6-29
Moving About a FOCUS File6-29
The SYSTEM Segment6-31
Segment Selection Qualification6-32
ERROR MESSAGES6-34
Status Return Codes (FCB(24)).6-34

Functions Performed by the HLI

The FOCUS HOST LANGUAGE INTERFACE (HLI) allows a programmer to access FOCUS files through 'calls' in his host program. These calls communicate with a FOCUS interface routine and permit:

- . Records to be retrieved sequentially.
- . Records to be retrieved in sort sequence.
- . Records to be retrieved based on a key value for which an index is available to provide rapid record location.
- . Records to be retrieved based on search criteria on any field in the record.
- . Field values in any record to be changed to new field values.
- . New records to be added to a file.
- . Existing records to be deleted from a file.

Program and HLI

On CMS the FOCUS HLI is incorporated into the user's program module at the time the module is generated. All communication to FOCUS files is accomplished via a 'call' to a subroutine named FOCUS. If the host program is coded in COBOL then the calling sequence is:

```
ENTER LINKAGE.  
CALL FOCUS USING ARG1 ARG2 ARG3 ARG4.  
ENTER COBOL.
```

In FORTRAN the calling sequence is:

```
CALL FOCUS (ARG1, ARG2, ARG3, ARG4)
```

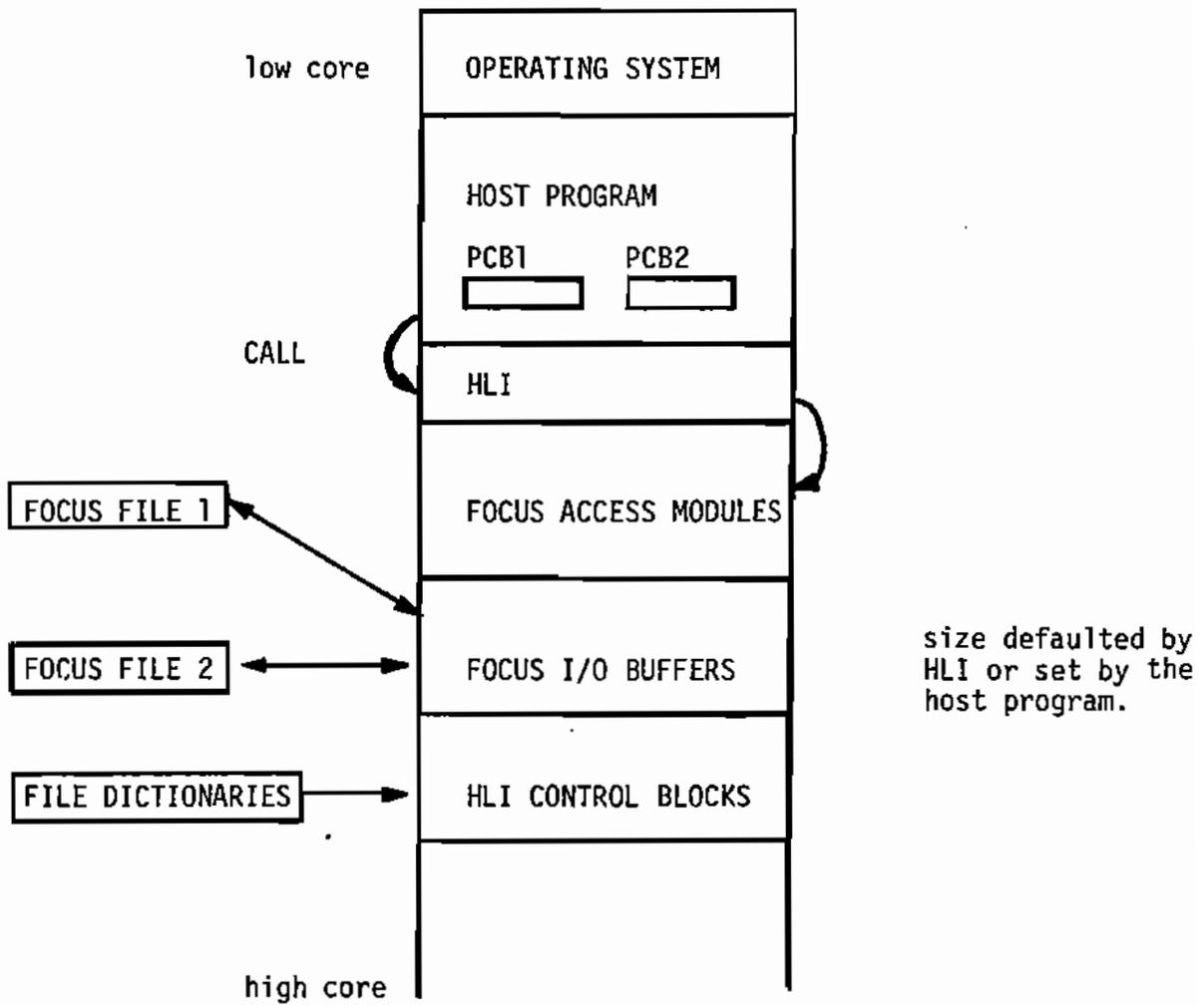
In PL-1 the assembler type calling sequence should be specified.

Constructing a Load Module:

A load module containing the user's own programs, plus the HLI supplied routines is generated by typing:

```
FOCUS GENHLI TEXT1 TEST2 ... TEXTn
```

Where TEXT_i are the names of the users text routines, a module having the same name as the first supplied text routine will be generated.



Core layout during HLI execution

Communicating with the HLI

The first argument in every call to FOCUS is the name of the command to be executed. The second argument in every call to FOCUS is the address of a communication work area which the host program provides. Both the host program and the HLI share this area. In the first part of this work area the host program places the name of the FOCUS data file to be processed, and then the name of the FOCUS dictionary containing the file description. Each time a call to the HLI is made, the result or status return code of the call is placed in this work area. Hence, by sharing this area, communication between the host program and the HLI is maintained.

One communication work area is required for each FOCUS file referenced directly by the host program. We will refer to this work area as the FILE COMMUNICATION BLOCK, or FCB.

The layout of an FCB area is:

<u>FCB Words</u>	<u>Item</u>	<u>Meaning</u>	<u>Bytes Used</u>
1-2	FN	CMS and FOCUS file name of data	8
3-4	FT	CMS filetype of data	8
5	FM	CMS filemode of data	2
6		Reserved	
7-8	DN	Dictionary filename if present must be 'MASTER' else blank	8
9-10	DT	Dictionary filetype	8
11	DM	Dictionary filemode	2
12		Reserved	
13-18		Reserved	
19-20	PASSCTL	File access password	8
21-22	NEWSEG	Name of highest new segment retrieved	8
23	SEGNUM	Segment number of NEWSEG (integer)	4
24	STATUS	Status return code (integer)	4
25	ERRORNUM	Detail of status if any (integer)	4
26-50		Reserved	

Note:

- A FCB is always 50 words, or 200 bytes in length.
- A different FCB is used for each open file, or each separate position in the same file.

Syntax of CALL to Subroutine FOCUS

Every call to the subroutine FOCUS contains from 2 to 9 arguments depending upon the subcommand to be executed.

There is a maximum of 9 arguments. A typical call with this many arguments might appear as:

```
CALL FOCUS ('NEX ', FCB, WKAREA, TARGET, ANCHOR, 0)
```

Meaning of the Calling Arguments

<u>Argument Name</u>	<u>Meaning</u>
WKAREA	A work area large enough for one retrieved record is the image of the 'SHO' command.
TARGET	The name of the target segment (8 bytes). This is the SEGNAME item in the file description.
ANCHOR	The name of the anchor segment (8 bytes). This is the SEGNAME item in the file description.
NTEST	The number of test conditions qualifying a retrieved record.
TESTREL	An array of 4 bytes for each field in the 'SHO' command which is either blank or contains the type of test relation. (EQ NE GE LE GT LT CO OM)
TESTLIT	A work area large enough for one retrieved record in the image of the 'SHO' command used to contain the literal values for qualifying record selections.
NUMB	Integer number used as an option by some commands.
SVAREA	An 8 word save area used to save information when an 'indexed' retrieval is requested (NXD). It is restored when the next record for the given index value is requested, and set to zero for the first occurrence.
NAMES	An area containing the names of the data fields which are to be activated. 12 bytes must be allocated for each fieldname.

Summary of HLI Commands

Command		
<u>Mnemonic</u>	<u>Name</u>	<u>Meaning</u>
OPN	Open	Open the FOCUS data file for usage.
SHO	Show	Establish a list of fields to be referenced.
NEX	Next	Retrieve next logical occurrence, qualified or unqualified, of target segment within parent segment.
CHA	Change	Change data values.
INP	Input	Include a new segment.
DEL	Delete	Delete a segment and all its descendents.
NXD	Next indexed	Locate a segment directly based on an indexed key value.
CLO	Close	Close the FOCUS file.
NXP	Next physical	Retrieve next physical segment, qualified or unqualified, of target segment.
FSP	First physical	Retrieve first physical segment, qualified or unqualified of target segment.
INFO	Information	Return file descriptive information such as fieldnames and segment names.
FST	First	Retrieve first logical occurrence, qualified or unqualified, of target segment within parent segment.

* All commands are 4 characters long. Leave a blank space after the 3 letter commands.

Command Syntax and Arguments on HLI Commands

NEX	FCB	WKAREA	TARGET	ANCHOR	NTEST	TESTREL	TESTLIT
FST	FCB	WKAREA	TARGET	ANCHOR	NTEST	TESTREL	TESTLIT
SHO	FCB	NAMES	NUMB				
OPN	FCB						
CLO	FCB						
CHA	FCB	WKAREA	TARGET	NULL	NUMB	TESTREL	TESTLIT
INP	FCB	WKAREA	TARGET	NUMB			
SAV	FCB						
DEL	FCB	TARGET					
INFO	FCB	WKAREA	NUMB				
NXP	FCB	WKAREA	TARGET	NTEST	TESTREL	TESTLIT	
FSP	FCB	WKAREA	TARGET	NTEST	TESTREL	TESTLIT	
NXD	FCB	WKAREA	TARGET	NTEST	TESTREL	TESTLIT	SVAREA

Record Work Area

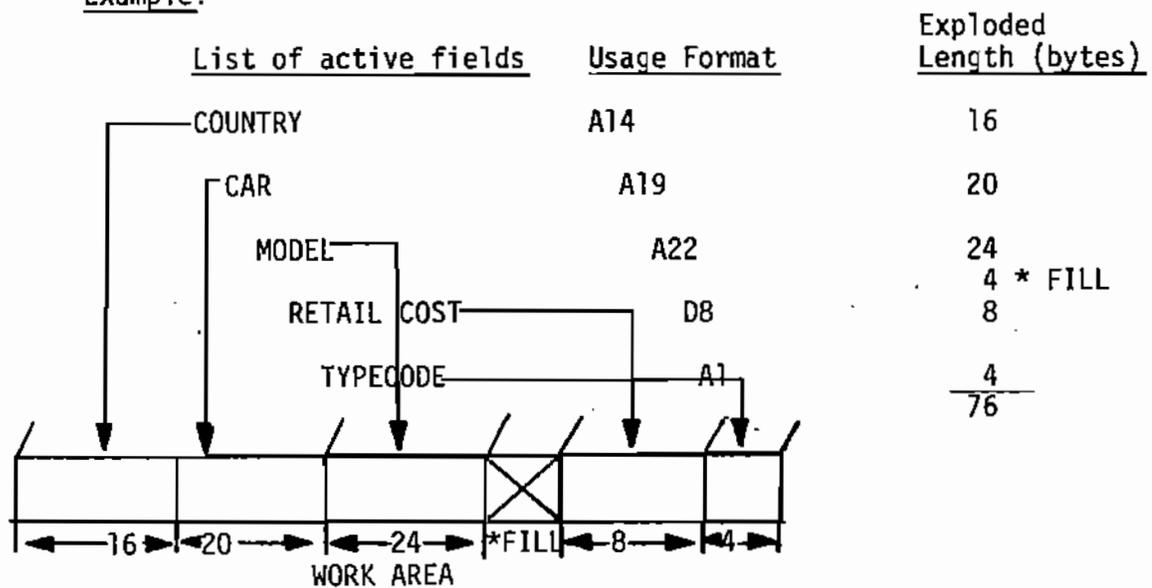
The host program must provide a work area into which retrieved records can be placed, or from which new data can be taken. The size and layout of this work area is controlled by the data fields which are activated by the SHO command.

The order in which the fields are provided in the SHO determines the order in which their data values are placed in the work area. The usage formats of the data fields determine the size of the work area. All lengths are rounded up to a multiple of 4 so that each data field starts on a full word boundary in the work area, or double word if the data is a double precision floating point number.

<u>Data Format Type</u>	<u>Length</u>
I	4 bytes
F	4 bytes
D	8 bytes
P	8 bytes
A n	$4 * [(n+3)/4]$

Note that a packed decimal number will always be returned in 8 bytes regardless of its usage length specification, e.g. P3, is padded out and returned in 8 bytes as if the format was P15.

Example:



Record Work Area (cont'd)

Because the host program specifies the fields and their order, it establishes a 'record image' through which all data transfer between it and the HLI occurs. This image will be oblivious to changes in the actual data file structure or layout, so long as the same field names are used. This is a valuable feature of the HLI and users should design their programs with this idea in mind so that their programs are independent of physical file changes.

HLI Command OPN (Open)

Syntax:

FORTRAN

```
CALL FOCUS ('OPN ', FCB)
```

COBOL

```
CALL FOCUS USING OPNCOMD FCB.
```

This must be the first call to FOCUS in a host program for each FOCUS file to be opened.

Action:

The dictionary and data file are located and made ready for subsequent usage. .

The data file must exist prior to usage by the HLI. It doesn't have to contain any data but it must have been initialized by FOCUS.

A status return code of zero means the file has been successfully opened. If the STATUS return code is non-zero, the file has not been successfully opened and any subsequent calls for its use will be rejected. A total of 20 FOCUS files and MASTER dictionaries may be open simultaneously.

After this command is successfully issued and before any 'SHO' command is issued all the fields in the file are considered on the 'SHO' list.

HLI Command SHO (Show)

Syntax:

FORTRAN

```
CALL FOCUS ('SHO', FCB, NAMES, NUMB)
```

COBOL

```
CALL FOCUS USING SHOCOMD FCB NAMES NUMB.
```

Action:

The list of names supplied in the area called NAMES controls the layout of all records retrieved. Each fieldname occupies 12 characters in the supplied list.

The order of the fields on the NAMES list sets the order in which fields are placed in the work area when data is retrieved. (See Record Work Area). When new file values are supplied for input or update operations they are taken from the work area.

In addition the same order is used when test conditions are supplied to qualify record selection.

Relation of SHO List to:

Test Relation List
Change Field Identity List

The area NAMES defines the order in which fields are referred to in test relation lists, and change value lists. The correspondence is illustrated in the example.

Example:

NAMES	TESTREL	CHA LIST	WKAREA or TESTLIT
COUNTRY	EQ		ITALY
CAR	EQ		FIAT
MODEL	EQ		128 2 DOOR AUTO
RCOST		EQ	3600
DCOST		EQ	3100
BODY		EQ	SEDAN
UNITS		EQ	46000

For qualifying purposes the area TESTREL sets the test relations by placing them in the corresponding position of the fields in the NAMES area. Hence, different sets of test relations, and/or test literals can be used by either changing the values in the areas, or supplying other areas in the different calls, i.e. TESTREL1, TESTREL2, etc.

When the change command is issued (CHA) the non-blank TESTREL entries indicates which fields are to be changed. Hence, different lists of changes can be accomplished by either supplying other CHALIST areas, i.e. CHALIST1, CHALIST2, etc., or replacing the blank, or non-blank values in one area.

Coding sequences of the following type are facilitated:

```

Locate
record → CALL FOCUS ('NEX', FCB, WKAREA, TARGET, ANCHOR, N, TESTREL, TESTLIT)
)test if found → IF (FCB(24).NE.0)GO TO XXX
change → CALL FOCUS ('CHA', FCB, WKAREA, TARGET, NUMB, CHAREL, CHALIT)
values
    
```

HLI Command NEX (Next)

Syntax: Without qualifying conditions on the retrieved segments;

FORTRAN

```
CALL FOCUS ('NEX ', FCB, WKAREA, TARGET, ANCHOR,  $\emptyset$ )
```

COBOL

```
CALL FOCUS USING NEXCOND FCB WKAREA TARGET ANCHOR ZERO .
```

with qualifying conditions on the retrieved segments;

FORTRAN

```
CALL FOCUS ('NEX ', FCB, WKAREA, TARGET, ANCHOR, NTEST, TESTREL, TESTLIT)
```

COBOL

```
CALL FOCUS USING NEXCOND FCB WKAREA TARGET ANCHOR NTEST TESTREL TESTLIT.
```

If issued in an unqualified form, the last argument must be zero, as this indicates that no test conditions are supplied.

The ANCHOR segment name may be 'SYSTEM' which is considered the parent of the root segment of the file .

HLI Command NEX (Next) (cont'd)

Action:

The next target segment is retrieved within the anchor segment (no change in anchor position), which meets the qualifying conditions, if any are provided. If a record is retrieved, the STATUS is 0. If no more target segments are available within the anchor segment, the status is 1. If any error occurred, the status is greater than 1. The retrieved segments are placed in the user supplied work area, WKAREA, in the image requested by the 'SHO' command.

HLI Command NXP (Next Physical)

Syntax: Unqualified

FORTRAN

```
CALL FOCUS ('NXP', FCB, WKAREA, TARGET, 0)
```

COBOL

```
CALL FOCUS USING NXPCOMD FCB WKAREA TARGET ZERO.
```

SYNTAX: Qualified

FORTRAN

```
CALL FOCUS ('NXP', FCB, WKAREA, TARGET, NTEST, TESTREL, TESTLIT)
```

COBOL

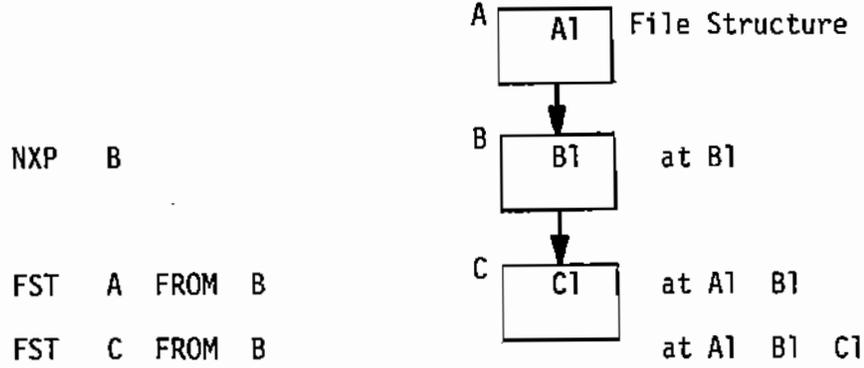
```
CALL FOCUS USING NXPCOMP FCB WKAREA TARGET NTEST TESTREL TESTLIT.
```

Action:

The next occurrence of the TARGET segment which is physically adjacent to the current one is retrieved. The segments are not retrieved in sort sequence. Only the target segment is retrieved. After the command is executed. The other parents and descendents can then be retrieved with FST commands.

HLI Command NXP (Next Physical) (cont'd)

Example: Connecting attached segments after either a physical or random retrieval of a target segment.



HLI Command NXD (Next thru index)

Syntax:

FORTRAN

```
CALL FOCUS ('NXD', FCB, WKAREA, TARGET, FIELD, NTEST, TESTREL, TESTLIT, SVAREA)
```

COBOL

```
CALL FOCUS USING NXDCOMD FCB WKAREA TARGET FIELD NTEST TESTREL TESTLIT SVAREA
```

Action:

The target segment is retrieved using the index of the field named in the FIELD argument. The value of NTEST must be at least one, and the literal value of the key must be in the TESTLIT area. The TESTREL area must have a relation of 'EQ' for the indexed field.

The first time the specific value of the key is provided the area called SVAREA must be all zeros. (SVAREA, 8 words). When the save area is zero the first occurrence of the key value is retrieved regardless of current position. Only the target segment is retrieved. It becomes the anchor point for obtaining related parent or descendent segments.

If there are multiple segments matching the key value then the next match is retrieved by reissuing the 'NXD' command, but with the same SVAREA as the last call. Do not zero out the SVAREA between calls as it contains the information necessary for processing all occurrences. When a new key value is to be supplied, then start with a zero'ed area.

There may be other qualifying test conditions on the retrieved target segment. These are provided in the TESTREL, and TESTLIT areas in the usual way.

HLI Command NXD (Next thru index) (cont'd)

Note : The argument 'FIELD' must be 12 characters and contain either the name or alias of the field whose index is to be used.

HLI Command CLO (Close)

Syntax:

FORTRAN

```
CALL FOCUS ('CLO', FCB)
```

COBOL

```
CALL FOCUS USING CLOCOMD, FCB.
```

Action:

The data file is removed from active usage. Any changes to the data are written to the disk and all buffer storage space is returned.

HLI Command CHA (Change)

Syntax:

FORTRAN

```
CALL FOCUS ('CHA', FCB, WKAREA, TARGET, NULL, NUMB, TESTREL, TESTLIT)
```

COBOL

```
CALL FOCUS USING CHACOND FCB WKAREA TARGET NULL NUMB TESTREL TESTLIT.
```

Action:

The data fields corresponding to the TESTREL areas which are non-blank (have an EQ), on the TARGET segment are changed to the values provided in the TESTLIT area. A copy of the after image of the segment changed is placed in WKAREA. Both TESTLIT, and WKAREA are in the SHO image record layout.

There are four bytes in the TESTREL area for each data field which was in the SHO list. The fields to be changed are indicated by the character 'EQ'. Only those fields which are on the named TARGET segment are changed.

Example:

	<u>SHO LIST</u>	<u>TESTREL</u>	<u>TESTLIT</u>
SEGNAME =MODREC	COUNTRY		
	CAR		
	MODEL		
	SEATS	EQ	5
	DEALER COST	EQ	3187

continued

HLI Command CHA (Change) (cont'd)

If the issued command is:

```
CALL FOCUS ('CHA', FCB, WKAREA, 'MODREC ', ' ', 2, TESTREL, TESTLIT)
```

then the two fields named SEATS, and DEALER COST will be changed to new values, which are provided in LITAREA.

HLI Command INP (Input)

Syntax:

FORTRAN

```
CALL FOCUS ('INP', FCB, WKAREA, TARGET, NUMB)
```

COBOL

```
CALL FOCUS USING INPCOMD FCB WKAREA TARGET NUMB.
```

Action:

The data in the WKAREA corresponding to the segment in TARGET creates a new segment in the file. Any values not provided on the segment are given default values of blanks if alphanumeric, and zeros if numeric.

The value of NUMB controls 2 options.

NUMB=0 insert segment after current segment

NUMB=1 insert segment before current segment

In neither case can a segment be inserted if it would break the sort sequence of existing segments. In this case the proper place for the segment will automatically be found regardless of the value of NUMB.

No descendent segments are included. Each of these must be added with a separate INP command.

The new segment becomes the current position.

HLI Command SAV (Save)

Syntax:

FORTRAN

```
CALL FOCUS ('SAV', FCB)
```

COBOL

```
CALL FOCUS USING SAVCOMD FCB.
```

Action:

All data values which have been changed via the CHA, DEL, or INP commands are written to disk storage. No modification to the current position occurs.

The command is useful in environments where there is a chance to lose a connection to the computer, and no copies of the transactions exist, hence, each one (or group) are written to the disk as they are processed, and not held in a buffer storage.

HLI Command DEL (Delete)

Syntax:

FORTRAN

```
CALL FOCUS ('DEL', FCB, TARGET)
```

COBOL

```
CALL FOCUS USING DELCOMD FCB TARGET.
```

Action:

The segment in TARGET is deleted from the file. All descendent segments from the deleted one are also deleted.

HLI Command FST (First)

Syntax: Without qualifying conditions on the retrieved segments;

FORTRAN

```
CALL FOCUS ('FST', FCB, WKAREA, TARGET, ANCHOR, 0)
```

COBOL

```
CALL FOCUS USING FSTCOMD FCB WKAREA TARGET ANCHOR ZERO.
```

Syntax With Qualifying Conditions

FORTRAN

```
CALL FOCUS ('FST', FCB, WKAREA, TARGET, ANCHOR, NTEST, TESTREL, TESTLIT)
```

COBOL

```
CALL FOCUS USING FSTCOMD FCB WKAREA TARGET ANCHOR NTEST TESTREL TESTLIT
```

If issued in an unqualified form the last argument must be zero, as this indicates that no test conditions are supplied.

If the anchor segment name is 'SYSTEM' i.e. FIRST TARGET FROM SYSTEM then the first logical target segment in the file will be retrieved.

)

HLI Command FST (First) (cont'd)

Action:

The first target segment is retrieved within the anchor segment (no change in anchor position) which meet the qualifying conditions, if any are provided. If a record is retrieved the STATUS (FCB (24)) is zero, if no record then it is 1 or some number greater than 1.

The target segment may be hierarchically below or above the anchor segment. When a segment is retrieved in a direct manner, from physical next, (NXP) or through an index (FND), then the balance of attached segments can be obtained via this command.

HLI Command INFO (Information)

Syntax:

FORTRAN

CALL FOCUS ('INFO', FCB, WKAREA, NUMB)

COBOL

CALL FOCUS USING INFOCMD FCB WKAREA NUMB.

Action:

Information about the location, length, and format of each field in the current 'SHO' command is returned.

Note: Upon opening the file ('OPN' Command) all fields by default are considered 'shown', hence by issuing the 'INFO' command immediately after the 'OPN' command complete file information is available.

The information returned in the WKAREA contains 12 words (48 bytes) for each field. The order of the fields is the order of the current 'SHO' command.

Layout of Return Work Area in INFO Command When NUMB=0

<u>Word</u>	<u>Meaning</u>
For each field {	1 Number of fields which follow.
	2-3 Name of segment.
	4-5-6 Name of data field.
	7-8-9 Alias name of data field.
	10-11 Usage format of data field.
	12 Length of field in bytes (binary integer).
	13 Starting byte position in current work area. (binary integer).

HLI Command INFO (Information) (cont'd)

Option Argument - NUMB:

When the value of NUMB is 1 the returned information is only about the segments and the file structure. This information is independent of the current SHO command.

Layout of Return Work Area in INFO Command When NUMB=1

	<u>Word</u>	<u>Meaning</u>
	1	Number of segments
	2-3	Name of segment
	4-5	Name of parent of segment
For	6	Starting byte in real segment of first field
each	7	Length of segment in bytes
segment	8	Segment type, alpha name
	9-10-11	Cross-reference key fieldname
	12	Number of sequence keys
	13-14-15-16-17	Reserved

The last argument in the INFO command should be either zero or one. Other values are reserved for other types of returned information which will be added in the future.

)

Moving About a FOCUS File

The basic command for moving from one record to another in a FOCUS file is the NEX command, or variants of it. A move starts at an 'anchor' segment and proceeds to a 'target' segment. All segments between and including the anchor and target are retrieved and placed in the supplied work area.

The move may be unqualified, without tests, in which case if a target segment exists the move is successful, or the move may be qualified, in which case a set of test conditions are supplied, and only if all of the test conditions are met are segments retrieved.

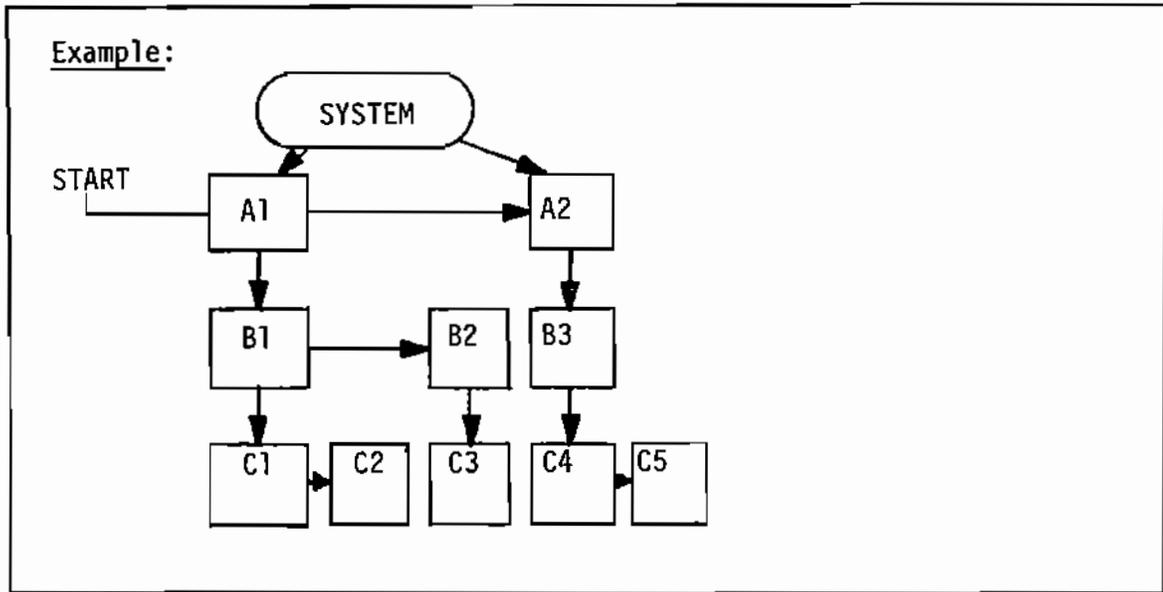
Selection of an anchor and target segment does not depend upon file structure hierarchy. Hence, a move can be:

- . from parent to descendent
- . from grandparent to grandchild (all levels of)
- . from child to parent
- . from grandchild to grandparent (all levels of)

Movement may be along physical paths, or keyed paths.

The only difference between physical paths and keyed paths is that only in physical files may data values be modified.

Moving About a FOCUS File (cont'd)

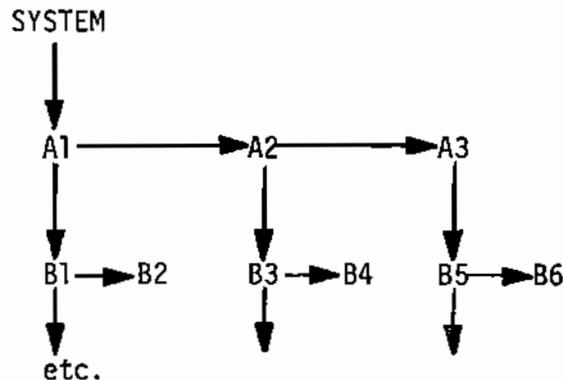


```

NEX FROM SYSTEM TO C retrieves A1 B1 C1
NEX FROM A TO C retrieves A1 B1 C1
NEX FROM A TO C no record.
NEX FROM A TO B retrieves A1 B2
NEX FROM SYSTEM TO A retrieves A2
NEX FROM A TO B retrieves A2 B3
NEX FROM B TO C retrieves B3 C4
NEX FROM B TO C retrieves B3 C5
    
```

The SYSTEM Segment

The notion that a chain of like segments are descendents of a unique occurrence of a parent segment is the basis of the NEXT LOGICAL command. It is so useful that this point of view is adopted also for chain of root segment occurrences: they are all viewed as descendents of a unique occurrence of a segment called SYSTEM, which is always current.



The SYSTEM segment contains no fields that can be retrieved or modified. It cannot be named as the target segment.

The command:

```
NEXT A FROM SYSTEM
```

if issued successively, will retrieve A1, A2, A3, etc., till end of file.

The command:

```
NEXT B FROM SYSTEM
```

will retrieve A1 and B2, then B2, then A2 and its first B descendents, etc...

Segment Selection Qualification

Qualifying conditions supported by the HLI are all of the form:

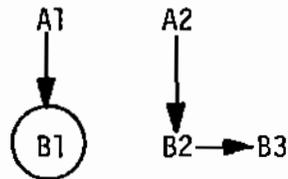
field relation value AND field relation value

The range of relation include all the more common relations such as equal, less than, etc... and others such as CONTAINS (CO) and OMITS (OM) appropriate for screening textual fields.

Since all the conditions that a path may have to satisfy are ANDed, the entire qualification fails as soon as any individual condition fails. The conditions are therefore not necessarily tested in the order specified, but rather segment by segment (but in the specified order within all conditions on the same segment) and as soon as a segment fails the path 'turns' toward the next possible target segment occurrence.

Specifically: when searching for a descendent target segment, the descendents of a segment occurrence which failed its conditions are not examined. Instead, the next segment retrieved will be the successor of its nearest ancestor, always within the limits of the search.

When the target is an ancestor segment the search is abandoned as soon as any condition fails:



Starting as above, the command NEXT A FROM B will retrieve A1 if all conditions are met, or else give up. A2 will not be considered because even if it met the necessary qualifications it would violate the definition of current position: to wit, that each current segment occurrence be a descendent of the current occurrence of its parent, if the latter exists.

Note that if A1 were current the command would be unsuccessful in any event, whether or not A1 met the qualifying conditions, because there would be no change in the current position.

)

Segment Selection Qualification (cont'd)

When the target segment is a descendent of the anchor segment, all current occurrences along the anchor-target path are tested to see if they meet the qualifying conditions.

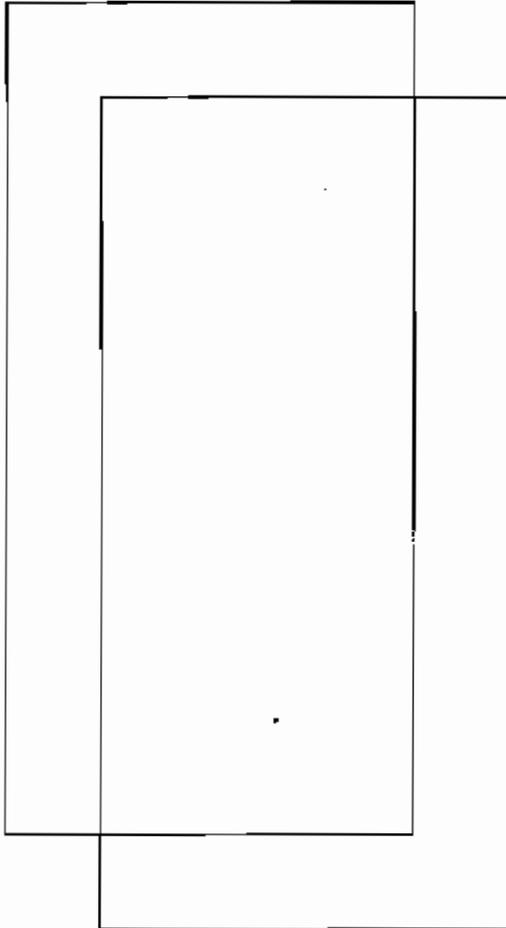
Host Language Interface
Status Return Codes (FCB(24))

<u>Return Code</u>	<u>Meaning</u>
0	Command executed normally.
1	Command executed normally, but no segments retrieved. The current position is unchanged. Either the qualifying conditions failed to locate desired record, or an end of chain condition occurred. (i.e. no more target segment within anchor segment).
760	Command not recognized.
761	Too few arguments in command. Syntax is therefore incorrect.
762	File not previously opened. i.e. first command was not OPN.
763	More core needed. Run cannot be continued.
764	MASTER file named as dictionary cannot be located.
765	Description of the data file not found in the MASTER dictionary.
766	No data found in the file. File must first be initialized through FOCUS MODIFY command even if no data is entered.
767	Error in file description encountered while opening file.
768	Parameter cannot be zero.
769	Fieldname is not recognized.
770	File wasn't opened, hence can't be closed.

Host Language Interface

Status Return Codes (FCB(24)) (cont'd)

<u>Return Code</u>	<u>Meaning</u>
771	Segment name not recognized.
772	Command or option recognized, but not supported yet.
773	No current position established from which to execute command.
774	Test relation is not recognized.
775	Attempt to change or improperly use a virtual keyed segment.
778	No path from anchor to target segment.
779	Error in use of indexed field, or field named in NXD does not have FIELDTYPE=I.
780	No value supplied for index lookup.
800	Serious error encountered which may be due to a problem in the operating environment.



F O C U S

USER'S MANUAL

INFORMATION BUILDERS INC.
254 WEST 31st STREET
NEW YORK, N.Y. 10001
(212) 736-4433

TABLE OF CONTENTS

Introduction.7-01
Naming FOCUS Command Files7-02
Executing FOCUS Command Files7-03
Procedures Without Missing Values.7-04
Procedures With Missing Values.7-05
Variable Names7-06
Supplying Values For Substitutable Variables7-07
Supplying Values on the EXEC Line.7-07
Continuing a Long EXEC Line.7-08
Rules for Supplying NAME=VALUE Sets7-08
Prompting for Substitutable Values7-09
Syntax for Substituted Variables7-10
Implied Prompting7-11
Direct Prompting7-12
System Variables7-13
Statistical Variables.7-14
Dialogue Manager Control Statements7-15
Concepts of Execution7-16
Testing and Branching.7-17
Labels.7-17
Unconditional GOTO.7-18
-GOTO label7-18
Conditional GOTO7-19
-IF Command7-19
Compound Expressions7-20
Testing the Length, Type, and Existence of a Variable7-21
-DEFAULTS Statement7-23
-TYPE Command7-24
Typing from a Labeled Line7-24
Assigning Values7-26
-SET Command.7-26
Ending a Stored Procedure7-27
-EXIT7-27
-QUIT7-27
-RUN7-28
Testing the result of a CMS command issued from FOCUS7-28
Automatically Executed Procedure, PROFILE, FOCEXEC7-30
Testing New Procedures7-31

Introduction

The purpose of the Dialogue Manager is to assist FOCUS users in executing procedures which are stored for repetitive use.

A stored procedure is a CMS file composed of FOCUS commands which may request a report or control a file maintenance operation. Instead of typing these commands on the terminal each time they are needed, the operator only types the name of the CMS file in which the commands are stored.

Parts of the stored procedure may be missing and require completion at the time of execution. This allows an operator to supply variable information and thus make a procedure more flexible. A typical use of this type would be a report request which sets up the format, heading, titles, etc. of a report but requires the operator to supply the screening conditions for record selection.

The process of easily obtaining these missing run time variables is one of the main features of the Dialogue Manager. As its name implies the emphasis is on conducting a dialogue with the operator. Specifically, the operator can be 'prompted' by a message typed on his terminal telling him what he has to do, then reading back his reply, and very importantly checking his reply for conformance with the instructions.

The missing variables can also be typed after the name of the procedure is typed, thereby removing the need for prompting to obtain them.

The Dialogue Manager provides facilities for setting default values for missing variables and for referencing 'system' variables such as the current date and time of day.

A procedure of FOCUS commands to be executed by the Dialogue Manager is created through the use of the CMS Editor. It is thereafter maintained in the same way.

Naming FOCUS Command Files

FOCUS procedures can be stored under any CMS filename and filetype. However, to remove the need to supply a filetype each time the procedure is executed and thus shorten the full name needed to identify the procedure the filetype of FOCEXEC is assumed if only a filename is provided.

For example, a CMS file named LOOKUP FOCEXEC will be assumed to be composed of FOCUS commands because the filetype is FOCEXEC.

The filemode, if not provided, is taken as '*', which means that the standard CMS disk search order is used, i.e. A disk, then B disk, etc. Therefore, stored FOCUS procedures may reside on central disks accessible simultaneously to many users.

Executing FOCUS Command Files

The Dialogue Manager is invoked from within FOCUS by typing the FOCUS command EXEC (or just EX) followed by the name of the procedure. i.e. EX procedure.

Example:

```
EX LOOKUP ← a procedure named  
LOOKUP FOCEXEC is executed
```

If the default filetype of FOCEXEC is not used then the full filename-filetype must be typed enclosed between single quotes because the full identifier contains an embedded blank. i.e. EX 'procname proctype'

Example:

```
EX 'DAILY REPORT' ← a procedure named  
DAILY REPORT is executed
```

Procedures Without Missing Values

A file composed of stored FOCUS commands without any missing values appears exactly as it would be if typed directly into FOCUS from the terminal.

Example: Stored Procedure

A CMS file named WEEKLY FOCEXEC consists of the following lines:

```
TABLE FILE CARS
"SUMMARY OF SALES...FOREIGN CARS"
SUM SALES AND PCT.SALES AND COLUMN-TOTAL
BY CAR BY MODEL
END
```

The report requested by this procedure is produced by entering FOCUS and typing:

```
EX WEEKLY
```

If the stored commands do not end the FOCUS procedure, that is, the terminator word, END, is not stored then the lines of the procedure up to the last line will be executed, but then the terminal will 'open' and allow the operator to complete the procedure 'live'. The operator thus has an opportunity to provide additional lines of variable information, or just type the word END.

Example: Partial Procedure - Operator supplies END

The stored procedure named COMP is:

```
DEFINE FILE CARS
MARGIN=100*(RCOST-DCOST)/DCOST ;
```

The procedure is executed as:

```
EX COMP
GAS-RATION=MPG/WEIGHT ;
END
```

← procedure invoked

← terminal opens

← operator adds a line

← operator completes procedure

Procedures With Missing Values

A file of FOCUS commands can be stored with variables which are to be supplied at the time of execution. The values which are to be so supplied are signaled by having an ampersand as their first character. For instance, in the phrase:

```
IF REGION IS &REGIONAME
```

the variable of REGIONAME is to be given a literal value at the time of execution.

The rules for naming substitutable variables are:

- . The first character preceding the name is an ampersand '&'.
- . The total number of characters in the name, excluding the ampersand is 12.
- . There are no embedded blanks in the name, or special characters of +, -, *, /, &.

Example: Procedure With Values to be Substituted

Procedure is named REPT15

```
SET MSG=OFF, PAGE=OFF
TABLE .FILE CARS
" TOTAL UNITS SOLD BY COUNTRY OF ORIGIN "
"     BODY TYPE IS &BODYTYPE "
SUM SALES AND PCT.SALES AND COLUMN-TOTAL
BY COUNTRY IF BODY IS &BODYTYPE
IF SEATS FROM &NUMSEATS
END
```

Note



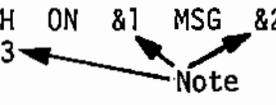
Positional Variable Names:

The substitutable variable may be assigned numbers in the order they are encountered and called simply &1, &2, &3, etc. A number may be between 1 and 255.

Example: Positional Variables

A procedure named NEWVAL is:

```
MODIFY FILE CARS
FIXFORM CAR/10 COUNTRY/10 MODEL/10 RCOST/6
MATCH CAR COUNTRY MODEL
ON MATCH UPDATE RCOST
ON NOMATCH REJECT
LOG NOMATCH ON &1 MSG &2
DATA ON &3
END
```



Note

SUBSTITUTABLE VARIABLES

- * Values typed on EXEC Line
- * Implied Prompting
- * Direct Prompting
- * System Variables
- * Statistical Variables

Supplying Values For Substitutable Variables

The Dialogue Manager provides two ways to supply values for substitutable variables.

- (1) Typed by the operator after the name of the procedure is typed.
- (2) Typed by the operator in response to 'prompts' for the missing values.

Supplying Values on the EXEC Line:

When the operator knows which values are required by a procedure, then they can be typed on the EXEC line along with the name of the procedure.

Example: Values for Named Variables

```
EX REPT15 BODYTYPE=SEDAN, NUMSEATS=4
```

If positional values are required then they are supplied either as 1=value, 2=value, or just a list of values with commas between them. i.e. value, value.

Example: Values for Positional Variables

```
EX NEWVAL    MORECS, OFF, NEWSTUFF
             ▲      ▲      ▲
             &1    &2    &3
```

When both positional and named variables are mixed together on the EXEC line they maintain the same interpretation. See the example below:

Example: Mixed Named and Positional Variables

```
EX COMPARE  MPG=30, COUPE, EAST, LIMIT=50
             ▲      ▲
             &1    &2
```

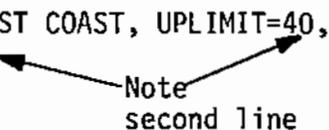
Continuing a Long EXEC Line:

When the list of directly supplied values exceed the width of the terminal then a comma should be the last character on the line, and the balance of the list typed on the next line.

Example: A Long List of Values

```
EX LOOKUP DIVISION=EAST COAST, UPLIMIT=40,  
LOWLIMIT=30, DEST=OFF
```

Note
second line



Rules for Supplying NAME=VALUE Sets:

Variables which are identified by their name require the name followed by an equal sign followed by the value to be assigned. Sets of NAME=VALUE's are separated from each other by a comma. i.e.

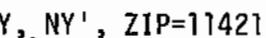
NAME=VALUE, NAME=VALUE, NAME=VALUE

The values to be supplied may be numbers, alphanumeric characters, etc., of any length. They may contain embedded blanks and special characters. However, if a value contains either an embedded comma, or embedded equal sign, then the value must be enclosed between single quotes.

Example: Value With Embedded Comma

```
EX NEWLIST ADDRESS='NY, NY', ZIP=11421
```

Embedded comma, note use of single quotes



A long list of name-value pairs may extend to the next line if the first line ends with a comma. However, a single name-value pair may not be split between lines.

Example: Two line response

```
EX LONGLIST ADDRESS='NEW YORK, NY',  
ZIP=14204
```

Prompting for Substitutable Values:

When a stored procedure is executed 'live' by the Dialogue Manager then any variable whose first character is an ampersand, and for which a value has not been supplied on the Exec line will be displayed on the terminal and a value requested from the operator. ('Live' operation is when the terminal is the source of input, i.e. is FILEDEF'ed as SYSIN). Hence, missing values will be automatically prompted.

There are two types of prompting which may be employed.

1. Implied Prompting

A missing value is encountered in the procedure at the point where its value is needed.

2. Direct Prompting

A Dialogue Manager command -PROMPT is used to obtain a value, and it is then used where needed in the procedure.

Syntax for Substituted Variables:

A substituted variable consists of three parts.

1. Name of variable
2. Length and type of reply value.
3. Text of prompt message.

Each of these parts is separated by a period from the next part. The format and text are optional.

```
&name.format.text.
```

1. Name of variable:

The name of a variable must be 12 characters or less. It is preceded by an ampersand. It may not contain the special characters of +, -, *, /, =.

Examples:

```
&LOCATION          prompts as LOCATION =
&COSTLIMIT        prompts as COSTLIMIT =
```

2. Format of variable:

The format is composed of two parts, a type and a length.

<u>Types</u>		<u>Length</u>
A	Alphanumeric	1 to 255 characters
I	Integer number	1 to 16 digits

Examples:

```
&COST.I5.        prompts as    COST =
                  The reply must be 5 or less digits.

&LOCATION.A3.     prompts as    LOCATION =
                  The reply must be 3 or less characters.
```

If a reply violates a format an error message is displayed, and the prompt re-issued. Note that the operator may terminate a procedure by replying QUIT to a prompt.

Syntax for Substituted Variables: (cont'd)

3. Text of prompt message:

If text is provided between periods after either the name or format, then the text is displayed instead of the variable name.

The text may itself contain a substitutable variable (name only).

Examples: Text of prompt message

```
&DIV.I4. ENTER FOUR DIGIT DIVISION NUMBER.
&LOC. LOCATION NAME IS.
&VALUE.I6. ENTER THE VALUE OF &VTYPE.
```

← Note no reply format

Implied Prompting:

When a substitutable variable has no value at the point where one is needed then the Dialogue Manager automatically requests a value from the terminal operator.

Example: Implied Prompting

The procedure is named WHATSUP

```
SET PRINT=&PRINT, MSG=OFF
TABLE FILE CARS
SUM SALES AND COLUMN-TOTAL
ACROSS BODY-TYPE
BY DEALER IF CAR IS &CARNAME
IF 'RETAIL COST' EXCEEDS &FROMCOST
END
```

The terminal session is:

```
ex whatsup
PLEASE SUPPLY VALUES REQUESTED
PRINT=offline
CARNAME=fiat
FROMCOST=3000
.
.
report follows.
```

← procedure is invoked

← no values supplied

← prompting needed

← first missing value

← second missing value

← third missing value

Implied Prompting: (cont'd)Example: Implied Prompting with Prompt Text

The procedure is named NEWMPG

```
MODIFY FILE CARS
* PROCEDURE TO UPDATE MPG
FIXFORM COUNTRY/10 CAR/10 MODEL/20 MPG/4
MATCH COUNTRY CAR MODEL
ON NOMATCH REJECT
LOG NOMATCH ON NONE MSG &MSG.MESSAGE ON OR OFF.
ON MATCH UPDATE MPG
DATA ON &NEWS.A8. ENTER NAME OF TRANSACTION FILE.
END
```

Direct Prompting:

Prompting does not have to be embedded in the procedure only at the point where a substitutable value is needed. Values can be requested before the point of need. This is accomplished by the Dialogue Manager command of -PROMPT. The syntax is:

```
-PROMPT &name.format. text.
```

Each direct prompt is a line by itself.

Example: Direct Prompting

```
-PROMPT &DIV.A10. ENTER NAME OF DIVISION=.
-PROMPT &DEST.A8. ONLINE OR OFFLINE PRINTING.
SET PRINT=&DEST
TABLE FILE SALES
SUM UNITS AND COLUMN-TOTAL
BY MONTH IF LOCATION IS &DIV
END
```

values obtained on
-PROMPT lines

System Variables

A list of variable names are reserved for system use. Values will automatically be substituted for them when they are encountered. There are two types of variables System Variables, and Statistical Variables.

<u>System Variables</u>	<u>Meaning</u>
&DATE	The current date. It is displayed as YY/MM/DD.
&TOD	The time of day. It is displayed as HH:MM:SS.
&MDY	The current date in MMDDYY form useful for numerical comparisons.
&ECHO	Test mode (See Testing New Procedures).
&DMY	Date in DD/MM/YY form.
&YMD	Date in YY/MM/DD form.

Statistical Variables

All of the 'statistics' of operation which can be displayed by the FOCUS command ? STAT are available for testing after FOCUS commands are executed.

<u>Statistical Variables</u>	<u>Meaning</u>
&LINES	Number of lines printed in last report.
&RECORDS	Number of records retrieved in last report.
&TRANS	Number of transactions processed in last MODIFY procedure.
&ACCEPTS	Number of transactions accepted in last MODIFY procedure.
&NOMATCH	Number of transactions rejected for not matching.
&FORMAT	Number of transactions rejected for a format error.
&INVALID	Number of transactions rejected because of an invalid condition.
&DUPLS	Number of transactions rejected as duplicates.
&INPUT	Number of segments input.
&CHNGD	Number of segments updated.
&DELTD	Number of segments deleted.
&RETCODE	A value returned by the operating system after a CMS command.
&BASEIO	Number of input/output operations.
&READS	Number of physical reads from external file.
&REJECTS	Number of other rejected transactions.

Dialogue Manager Control Statements

A stored procedure may be composed of two types of lines. Those which are to be passed to FOCUS, and are FOCUS commands, and those which are used only by the Dialogue Manager. Dialogue Manager control statements always begin with a dash '-' as their first non-blank character in positions 1 or 2 of a line.

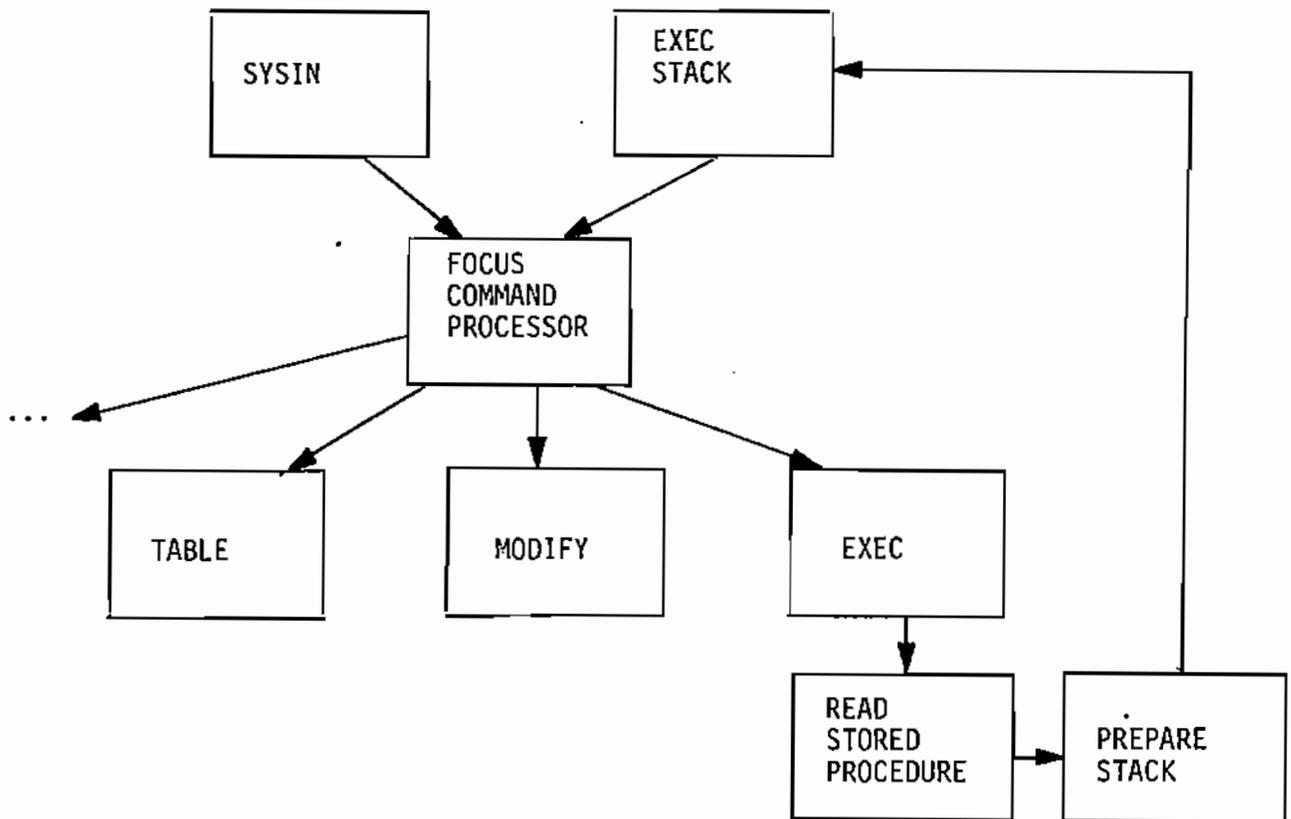
Dialogue Manager Control Statements

Control Statements	Meaning
-*	comment line, has no action
-DEFAULTS	sets initial values for substituted variables
-EXIT	exit and execute the FOCUS commands
-GOTO	unconditional branch
-IF	test and branch statement can use
-label	label which a branch statement can use
-PROMPT	types a message and reads a reply
-QUIT	exit but do not execute procedure
-RUN	execute the FOCUS commands to this point, then return
-SET	assign a value to a variable
-TYPE	types a message
-TYPE1	types a message after issuing page eject
-TYPE0	types a message after skipping a line
-TYPE4	types a message but doesn't 'line feed' after

Concepts of Execution

The Dialogue Manager reads the lines of a stored procedure, expands them with the values of variables which have been supplied at run time and stacks them for execution by FOCUS. Only the lines beginning with a dash in columns 1 or 2 are not stacked. These lines are assumed to be Dialogue Manager control statements, or labels. Control statements are executed immediately as their purpose is to manage the stacking of lines. When a procedure exits all of the lines which have been stacked are then passed to the regular FOCUS command processor as an alternate input stream. FOCUS will read from this stack until it is exhausted, then either return to reading the regular input stream which is FILEDEF'ed as SYSIN or return to the EXEC processor. If a -RUN control command was the cause of exit from the procedure and FOCUS is ready to read a new command, i.e. is at the command level, then the procedure is re-entered at the point after the RUN statement.

Schematically the process is shown below:



Dialogue Manager 'Stacking' Process

Testing and BranchingLabels

A label is a Dialogue Manager line beginning with a dash in position 1 or 2 and followed by a name of 12 or less characters. A branching statement will search for a label it needs by reading forwards. If the label is not found, then a circuit will be executed, reading again from the top of the stored procedure. Hence, a branch can be to a label which has already been passed, thus enabling a looping ability in stored procedures.

Example: Decision Based on Substituted Value

```

-PROMPT &KIND. 'DO YOU WANT DETAIL OR SUMMARY'.
test of → -IF &KIND EQ 'DETAIL' GOTO DETREP ;
reply      TABLE FILE CARS
           "SUMMARY REPORT &DATE"
           SUM INVENTORY AND COUNT BY CAR
           END
           -EXIT
alternate → -DETREP
report     TABLE FILE CARS
           "DETAIL REPORT &DATE"
           PRINT INVENTORY BY CAR BY DEALER
           END

```

Example: Testing the Results of a Prior Command

```

MODIFY FILE CARS
FIXFORM CAR/10 X1 COUNTRY/10 XL MODEL/2 BODY
FIXFORM DEALER/10 INVENTORY/5
MATCH CAR COUNTRY MODEL DEALER BODY
ON MATCH UPDATE INVENTORY
ON NOMATCH REJECT
LOG REJECT ON NOVALS MSG OFF
DATA ON NEWINV
END
Note → -RUN
test of → -IF &TRANS EQ 0
result  -OR IF &NOMATCH/&TRANS LT .10 GOTO EXIT ;
        TABLE FILE CARS
        "DEALER RECORD AS OF &DATE "
        LIST DEALER AND CAR AND MODEL AND BODY
        AND INVENTORY
        END
        -EXIT

```

Unconditional GOTO-GOTO label

An unconditional branch is a Dialogue Manager statement beginning with -GOTO. The syntax is:

```
-GOTO label
```

The target label may be further on in the procedure, or may be prior to the branch, in which case the branch is a backwards branch. If the label cannot be found it is an error and the stored procedure is not executed.

Example: Unconditional Branch

```
-TYPE WHEN FINISHED TYPE QUIT
-START
-PROMPT &SCREW. PRODUCT NUMBER=.
TABLE FILE PARTS
"PRODUCT NAME PNAME ON &DATE"
"LOCATION LOG "
"INVENTORY INV "
IF PRODCODE IS &SCREW
END
-GOTO START
```

Conditional GOTO-IF Command

The sequential execution of one Dialogue Manager statement after another can be altered by use of 'branching' statements. A branch is a line which begins with -IF. The syntax is:

```
-IF expression GOTO label [ELSE GOTO label];
```

The ELSE clause is optional; if not present then the line following the branch is the default ELSE location. (As if the statement was ELSE CONTINUE).

The expression to be tested may use all of the arithmetic and logical operations and special function available. (See Report Preparation-Calculations). The result of the expression is either true or false. If true then the print 'GOTO' is invoked, else if false the second GOTO is invoked.

A branching statement may occupy as many lines as needed, but the end of the statement must be signaled by a semi-colon.

Usage:

Branching can be used to test the values of substituted variables and then select the FOCUS proceedings to be executed. Similarly, the internal system statement variables can be tested after a FOCUS procedure and based on these results other procedures executed. For example, if too many records (&RECORDS) are retrieved in one report a diagnostic report can be run.

Conditional GOTO (cont'd)Compound Expressions:

An expression may consist of multiple IF-THEN parts each with a 'label' as the result. i.e.

```
-IF &COST GT 10000 GOTO HIGH ELSE IF &MPG  
-LT 30 GOTO MID ELSE GOTO REG ;
```

Note that a continuation line of a long expression starts with a dash.

Example: An Illustration of Conditional Branching

```
-* PROCEDURE TO MODIFY FILE CARS  
-* TRANSACTIONS IN FREEFORM, FIXFORM, OR THROUGH PROMPT  
-DEFAULT &ONFILE=' '  
CMS FILEDEF DISKFILE DISK &FN.CMS FILENAME. &FT.CMS FILETYPE.  
MODIFY FILE CARS  
-ASKUSER  
-IF &FORM. ENTER DATA FORMAT - FIXED, FREE OR PROMPT. IS FIXED  
- GOTO FIXFORM ELSE IF &FORM IS FREE GOTO FREEFORM ELSE  
- IF &FORM IS PROMPT GOTO PT ELSE GOTO ASKUSER ;  
-FIXFORM  
-SET &ONFILE='ON DISKFILE' ;  
FIXFORM CAR/6 COUNTRY/8 MODEL/30  
-GOTO THEREST  
-FREEFORM  
FREEFORM CAR COUNTRY MODEL  
-GOTO THEREST  
-PT  
PROMPT CAR COUNTRY MODEL  
-THEREST  
MATCH CAR COUNTRY  
ON MATCH REJECT  
ON NOMATCH INCLUDE  
DATA &ONFILE  
-IF &FORM NE FIXED GOTO EXIT ;  
END  
-EXIT
```

Testing the Length, Type, and Existence of a Variable

In a procedure in which the branching process is involved, or some of the variables must be supplied on the initial EXEC line, then it is necessary to be able to test if any values were ever supplied for a variable, and if so, what type of value it has, numerical or non-numerical, and the length of the value. For example, it would be an error to perform a numerical computation on a variable for which non-numerical data values were supplied.

Three tests functions are available. They are each placed immediately after the name of the variable to be tested.

Existence of value:

SYNTAX

`&name.EXIST`

The value is a zero if no value has ever been supplied for the variable, and non-zero otherwise.

Example:

```
-IF &RCOST.EXIST EQ 0 GOTO ASKFOR  
-ELSE GOTO HAVE ;
```

Length of Value:

SYNTAX

`&name.LENGTH`

The value is the number of characters forming the current length of the variable, or zero if no value has been supplied.

Example:

```
-IF &DIVISION.LENGTH GT 15 GOTO NOTGOOD  
-ELSE GOTO OK ;
```

Testing the Length, Type, and Existence of a Variable (cont'd)

Type of Value:

SYNTAX

&name.TYPE

The result is 'A' if any non-numerical characters form the current value, else the result is 'N' if the current value is a valid number.

Example:

```
-IF &DCOST.TYPE NE 'N' GOTO NOGOOD ;
```

)

DIALOGUE MANAGER CONTROL STATEMENTS

)

- * Concepts

- * Statements

-DEFAULTS Statement:

Generally, unless some value is supplied for a substituted variable, the FOCUS command will be invalid if it is executed. When the procedure is not executed 'live', and the list of values is to be supplied on the EXEC line then variables for which no value is provided will be blank. Hence, if a line for instance contains the phrase IF DIVISION IS &DIV and no value for &DIV is available, the phrase IF DIVISION IS is invalid and the FOCUS procedure will not be executed.

The -DEFAULTS control allows default values to be supplied for substitution variables. If any value is also provided on the EXEC line then the default is not used. Values are identified by name and multiple sets of name-value pairs are typed on the -DEFAULTS line. The syntax is:

```
-DEFAULTS &name=value, &name=value, &name=value
```

Example: Default Values

```
-* PROCEDURE NAMED SUMMARY  
-DEFAULTS &LOCATION=$$$$$$, &LOWER=0  
TABLE FILE SALES  
SUM UNITS ACROSS MONTH  
BY TYPE IF AREA IS &LOCATION  
IF VALUE EXCEEDS &LOWER  
END
```

If the procedure is executed as:

```
EX SUMMARY LOWER=30
```

Then the variables have the values of

```
&LOCATION=$$$$$$ (masks all values)  
&LOWER=30
```

-TYPE Command:

Messages are typed from a stored procedure on Dialogue Manager lines beginning with the word -TYPE.

```
-TYPE text
```

Substitutable variables may be embedded in the text. i.e.

```
-TYPE REPORT FOR &REGION ON &DATE
```

Three alternate forms of -TYPE are available which pass the printer control character to the output device.

```
-TYPE+ text
```

A line feed following the printing of the text is suppressed.

```
-TYPE0 text
```

A line is shipped before the current text is displayed.

```
-TYPE1 text
```

A page eject occurs before the line is printed.

Typing from a Labeled Line:

A TYPE statement may be issued on a line which starts with a label. The syntax is:

```
-label TYPE text
```

Typing from a Labeled Line: (cont'd)

Example: Typing from a labeled line

```
-IF &VALUE GT 1000 GOTO NOGO ;  
-TYPE OK...  
EXEC REPT1  
-EXIT  
-NOGO TYPE VALUE IS TOO LOW  
-EXIT
```

Assigning Values

-SET Command

Substitutable variables may be assigned values which are computed in arithmetic or logical expressions. The syntax is:

```
-SET &NAME=expression ;
```

Note the semi-colon at the end of the expression. It is needed as an expression may occupy several lines.

Example: Transforming an initial value

```
-TYPE DO YOU WANT POSITIVE VALUES  
-PROMPT &YN.ENTER YES OR NO.  
-SET &VAL=IF &YN EQ 'YES' THEN 1 ELSE 0 ;  
TABLE FILE POINTS  
SUM NUMBER BY CLASS  
IF POSITION IS &VAL  
END
```

Example: Controlling a loop

```
-DEFAULT &N=0  
-START  
-SET &N=N+1 ;  
-PROMPT &OK.ANOTHER VALUE? Y/N.  
-IF &OK EQ 'N' GOTO -EXIT ;  
-IF &N GT 5 GOTO -NOMORE ;  
EXEC REPORT  
-RUN  
-GOTO START ;  
-NOMORE TYPE EXCEEDING REPETITION LIMIT  
-EXIT
```

Ending a Stored Procedure

-EXIT

When the label -EXIT is encountered processing of the lines in the procedure is ended and execution of the expanded procedure lines begins.

If the last line of the procedure is reached and no more lines are present then an implied -EXIT occurs.

The -EXIT label is useful for ending a procedure at different points when branching statements are used. The following example illustrates this:

Example: Use of -EXIT

```

        PROMPT &WHERE.DETAILED OR SUMMARY REPLY DorS.
        -IF &WHERE EQ 'D' GOTO TWO ;
        TABLE FILE CARS
        SUM SALES BY CAR
        END
Note → -EXIT
position -TWO
        TABLE FILE CARS
        PRINT SALES AND MODEL BY CAR
        END
```

-QUIT

The label -QUIT causes an immediate exit from the stored procedure, and the lines which have been stacked for execution are not executed. (If the response to any prompt is QUIT it has the same effect as encountering a -QUIT control statement).

Ending a Stored Procedure (cont'd)-RUN

The label -RUN causes an immediate exit from the stored procedure. The stacked lines are executed, and when they are exhausted the processing of the stored procedure then resumes at the line after -RUN.

This is an important facility because it enables the results of a FOCUS command such as TABLE, or MODIFY, etc, to be tested and a branch taken based on the result; for instance if the statistical variable &RECORDS is to be tested after a report executes. This variable is the number of records retrieved for a TABLE request statement. The following example illustrates this.

Example: The -RUN statement

```
-START
TABLE FILE FCAR
"COMPARISON OF MILEAGE"
PRINT MODEL AND BODY BY CAR
IF MPG GE &MPG.I2
IF WEIGHT LE &WEIGHT.I4.
END
-RUN
-IF &RECORDS EQ 0 GOTO RETRY ;
-EXIT
-RETRY
-SET &MPG=&MPG-10 ;
-IF &MPG LE 0 GOTO EXIT ELSE GOTO START ;
-EXIT
```

Testing the result of a CMS command issued from FOCUS:

The statistical variable &RETCODE is the value returned by the operating system after a CMS command is executed. These commands are issued by typing CMS, followed on the same line by the text of the command i.e.

CMS LISTF * MASTER *

Testing the result of a CMS command issued from FOCUS: (cont'd)

Generally, if the command executes without error the value of &RETCODE will be zero. (See the CMS Command Language Guide). The following example illustrates its usage.

Example: &RETCODE

```
CMS STATE MYTRANS DATA A1
-RUN
-IF &RETCODE NE 0 GOTO BAD ;
CMS FILEDEF INDATA DISK MYTRANS DATA A1
MODIFY FILE FCAR.
MATCH COUNTRY CAR MODEL BODY
ON MATCH UPDATE RCOST
ON NOMATCH REJECT
DATA ON INDATA
END
-EXIT
-BAD TYPE TRANSACTION FILE NOT AVAILABLE
-EXIT
```

SPECIAL TOPICS

- * PROFILE FOCEXEC
- * Testing New Procedures

PROFILE FOCEXECAutomatically Executed Procedure

If a file named PROFILE FOCEXEC A1 is present on a CMS disk it will automatically be executed each time FOCUS is entered 'live'. This is useful for setting environment parameters such as issuing FOCUS USE and SET commands, and CMS FILEDEF's. Complex DEFINE'd variables could be automatically computed and available.

Example: PROFILE FOCEXEC

```
USE
MASTER FOCUS C1
END
SET PAUSE=ON, MSG=OFF
CMS FILEDEF MYSAV DISK SAVE TEMP (LRECL 304 RECFM V)
DEFINE FILE CARS
SPREAD/I6=IF RCOST GTO THEN (RCOST-DCOST)/DCOST
           ELSE 1.30*DCOST ;
END
-TYPE FOCUS SESSION ON &DATE AT &TOD
```

Upon entering FOCUS the procedure is executed and the message displayed;

```
FOCUS SESSION ON 06/09/76 AT 12:21:06
```

Testing New Procedures

The basic function of the Dialogue Manager is to expand stored FOCUS procedure lines which may or may not contain substitutable variables and pass these lines to FOCUS. Other lines encountered in the procedure which begin with a dash are used by the Dialogue Manager itself. They are interpreted as Dialogue Manager commands. These are not passed to FOCUS.

In order to display and test a new procedure an ECHO mode is available which is set by assigning a value to a variable named &ECHO. It can have three values.

```
          ON  
&ECHO= OFF (default)  
          ALL
```

The value is assigned either in a -DEFAULTS command, or on the EXEC line. For example,

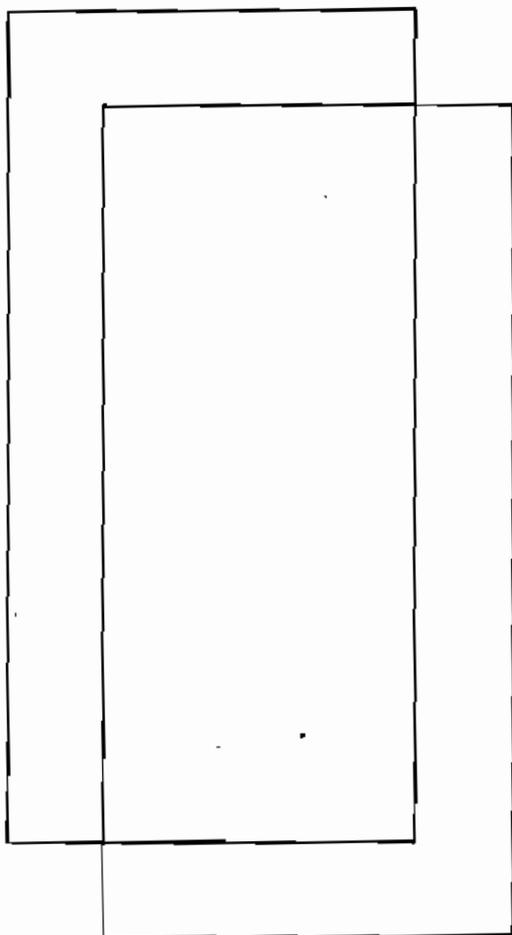
```
EX SUMMARY ECHO=ON
```

or

```
-DEFAULT ECHO=ALL
```

The effect of these values are:

- | | |
|----------|--|
| ECHO=ON | Lines which are expanded and would be stacked for FOCUS execution are displayed on the terminal instead. FOCUS then executes the lines. |
| ECHO=ALL | Dialogue manager control lines, as well as lines which are expanded and would be stacked for FOCUS execution are displayed on the terminal instead. FOCUS then executes the lines. |
| ECHO=OFF | The normal default. Stacked lines are not displayed, and are executed. |



F O C U S

USER'S MANUAL

INFORMATION BUILDERS INC.
254 WEST 31st STREET
NEW YORK, N.Y. 10001
(212) 736-4433

(FOC001) THE NAME OF THE FILE IS MISSING.

The request statement is missing the word FILE, or data field names are encountered before the identity of the file is provided.

(FOC002) A WORD IS NOT RECOGNIZED.

A word which is not a reserved word begins a phrase. A reserved word may have been misspelled or there is a syntax error. Reserved words are IF, BY, AND, etc.

(FOC003) THE FIELD NAME IS NOT IN THE DICTIONARY.

A word which is assumed to be the name of a data field does not appear on the list of names or aliases for the file.

(FOC004) THE OPTION ON THE VERB OBJECT IS NOT RECOGNIZED.

The prefix in front of the data field name is not valid. It is not either PCT*, AVE*, MAX*, MIN*, FST*, LST*, ASQ*, or TOT*.

(FOC005) THE NUMBER OF VERB OBJECT FIELDS EXCEEDS MAXIMUM.

There are more than 64 fields mentioned explicitly in the verb phrase.

(FOC006) FORMAT OF TEST VALUE IS INCORRECT.

The value supplied as a literal violates the format rule for the data field to be tested. For example, the data type is computational but a non-digit appears in the literal.

(FOC007) REQUEST STATEMENT DOES NOT CONTAIN A VERB.

The request statement does not contain a verb, and the heading, if any, does not contain a reference to any data fields, implying a verb.

(FOC008) NUMBER OF IN-GROUPS-OF FIELDS EXCEEDS 5.

The number of sort fields grouped into ranges exceeds five.

- (FOC009) INCOMPLETE REQUEST STATEMENT.
The request statement is terminated with an END, RUN, semi-colon (;), or end-of-file before it is completed.
- (FOC010) THE NUMBER OF SORT CONTROL FIELDS..BY..EXCEEDS MAXIMUM.
There are more than 64 BY phrases in a single request statement.
- (FOC011) THE TEST RELATION IS NOT RECOGNIZED.
The test relation in an IF phrase is not a valid type of test such as IS, FROM, TO, etc.
- (FOC012) THE WORD 'FILE' APPEARS TWICE.
The word FILE appears more than once in the request statement. A file has already been identified.
- (FOC013) THE 'ON' CONDITION FIELD IS NOT ALSO A SORT CONTROL FIELD.
The ON condition requires that the field named also be used in a BY phrase as a sort break but the BY phrase is absent.
- (FOC014) THE NUMBER OF 'ACROSS' SORT CONTROL FIELDS EXCEEDS MAXIMUM.
There are more than 5 ACROSS phrases in a single request statement.
- (FOC015) TEST VALUE HAS MORE CHARACTERS THAN FIELD FORMAT LENGTH.
An alphanumeric data field is being compared to a test value which has more characters than the format of the field allows.
- (FOC016) THE TRUNCATED NAME USED FOR THE FIELD IS AMBIGUOUS.
The truncated set of characters used instead of the full field name or field alias is not unique. For example, if two fields are named 'COST' and 'COMPANY' the representation by the letters 'CO' is ambiguous. Use more letters in the name or the full name.
- (FOC017) THE NUMBER OF TEST CONDITIONS EXCEEDS MAXIMUM.
The number of literal test conditions in 'IF' type of phrases exceed the maximum capacity to store the literal data. About 3000 characters are available.

(FOC018) THE OPTION IN THE 'ON' PHRASE IS INVALID.

The actions named after the field in the ON phrase is not one of the valid options such as SKIP-LINE, SUB-TOTAL, etc.

(FOC019) THE NUMBER OF SETS OF VERBS EXCEEDS MAXIMUM.

the number of sets of sort fields each with their own verb objects exceeds 6. This occurs only for multiple requests which are combined in the same statement. e.g. SUM A BY B BY C SUM D BY A...etc.

(FOC020) THE SETS OF SORT CONDITIONS ARE NOT RELATED.

In a request statement with multiple sets of different verb objects associated each with their own sort conditions, the conditions are not logically related. e.g. SUM A BY B BY C SUM D BY C

(FOC021) MAXIMUM HEADING/FOOTING SIZE EXCEEDED.

The free text supplied for the heading and/or footing exceed the allowed maximum. Possibly the use of double quotes at the start and end of each line will reduce the number of trailing blanks stored as free text.

(FOC022) MAXIMUM REPORT TITLES SIZES EXCEEDED.

Each displayed data field may have from 1 to 5 lines as a column title, although the test for each column is not specifically limited; in total about 800 characters of titles can be provided.

(FOC023) MULTIPLE SET OF 'BY' FIELDS MUST INCREASE IN NUMBER.

Multiple verbs are present and the sort control fields for a set do not contain at least all of the same ones from the prior set, e.g. the following is an error: SUM A BY B BY C SUM D BY B

(FOC024) LIST OR PRINT MUST BE IN LAST SET IF MULTIPLE SETS.

The verb LIST or PRINT can only be used in the last set of multiple set of verbs, e.g. correct form is SUM A BY B LIST D BY B.

(FOC025) IN-GROUPS-OF FIELD IS NOT NUMERICAL

The test literal for the IN-GROUPS-OF test is not a valid number.

(FOC026) REPORT NOT AVAILABLE FOR RETYPING.

A command such as RETYPE, HOLD, or SAVE has been issued which requires that a TABLE command be the immediately prior command. There is no TABLE report currently available.

(FOC027) FIELD IS NOT NUMERICAL...CANNOT USE IN-GROUPS-OF.

The data field is not defined as a computational number, hence, it cannot be grouped into ranges.

(FOC028) INCLUDES OR EXCLUDES TEST CANNOT BE USED WITH EXTERNAL FILE.

The test relations of INCLUDES or EXCLUDES can only be applied to FOCUS files where the order of occurrence of record segments can be controlled.

(FOC029) SORT KEYS NOT IN SINGLE TOP-TO-BOTTOM SEGMENT PATH.

The request statement contains directions to sort the records retrieved in an illogical manner.

(FOC030) SORT KEYS NOT IN PATH OF ALL VERB OBJECTS IN VERB SET.

The request statement contains an illogical sort condition relative to the field to be retrieved and sorted. A multi-set request statement where each set of verb objects have their own sort keys may be appropriate.

(FOC031) THE NUMBER OF VALUES RETRIEVED FOR ACROSS EXCEEDS 36.

The number of values retrieved for the ACROSS field exceeds 36. Retrieval may not be complete but an attempt to print the report is made. If it cannot be printed use the word 'BY' instead of 'ACROSS'.

(FOC032) RECTYPE FIELD MISSING FROM MUTI-RECORD FILE DESCRIPTION.

In an external (non-FOCUS) file description, one of the record segments is missing a needed identification field named RECTYPE.

(FOC033) THE TOTAL TEST IS NOT APPLIED TO A VERB OBJECT FIELD:

The test condition IF TOTAL filename...does not refer to a data field which is being accumulated. It must refer to a verb object field which is used in the request statement.

(FOC034) THE PRINT LINE EXCEEDS THE MAX WIDTH OF THE PAGE:

The number of characters in the print line exceeds the maximum page width and therefore cannot be displayed. Automatic compression to 1 space between fields has been tried in an effort to fit the report on the page.

Change the request statement to use the options of OVER, or FOLD-LINE, or reduce the width of the columns by making the titles smaller.

(FOC035) ERROR IN FORMAT OF COMPUTED FIELD DEFINITION:

A computed field has a format which is not valid. Check the left-hand side of the expression before the equal sign.

(FOC036) NO DATA FOUND FOR FOCUS FILE NAMED:

A report request is valid but no data exists for the file. Perhaps a USE command is needed if the default filetype and filemode have been changed.

(FOC037) REQUEST IS INCOMPATIBLE WITH TABLEF (EX:ACROSS):

The TABLEF command prints records as they are retrieved hence operations which request a complete report are not logical. For instance, ACROSS, or PCT.field.

(FOC038) EXTERNAL FILE DOES NOT CONTAIN ANY TEST LITERALS:

The request refers to an external file for a list of literal test conditions. This file is empty, and no other list or separate items are provided to test against.

(FOC201) INTERRUPT .. DIVISION BY ZERO.

An attempt to perform an arithmetic operation. Non-arithmetic data has occurred. Error cause is indeterminate and possibly due to input of binary data which bypasses the normal character checking operation.

(FOC202) INTERRUPT .. FLOATING VALUE OVERFLOW.

An attempt to perform an arithmetic operation. Non-arithmetic data has occurred. Error cause is indeterminate and possibly due to input of binary data which bypasses the normal character checking operation.

(FOC203) INTERRUPT .. FLOATING VALUE UNDERFLOW.

An attempt to perform an arithmetic operation. Non-arithmetic data has occurred. Error cause is indeterminate and possibly due to input of binary data which by passes the normal character checking operation.

(FOC204) TOO MANY INTERRUPTS .. RUN TERMINATED.

Errors 1, 2, and 3 have occurred more than ten times. Run is terminated based on a maximum error count.

(FOC205) THE FILE NAMED IS NOT IN THE DICTIONARY.

The data file which is referenced cannot be found in the FOCUS Master Dictionary or among the names of external files in the Master Dictionary.

(FOC206) INSUFFICIENT CORE SPACE IS AVAILABLE FOR LOADING PROGRAM.

The amount of core storage is insufficient for loading the program or while in execution a request for a user written program has been processed and there is insufficient space for it. Define larger storage and retry procedure.

(FOC207) ERROR IN FORMAT DEFINITION OF FIELD

The format of a data field in the Master Dictionary has a format which is not a recognized type or length or has invalid edit options.

(FOC208) NUMBER OF FIELDS EXCEEDS 256.

The maximum number of data fields in the file and the number of temporary data fields which is defined exceed 256.

(FOC209) THE DATA VALUE EXCEEDS THE LENGTH SPECIFIED FOR IT.

The number of characters obtained for an alphanumeric data field is greater than the length specification in the format.

(FOC210) THE DATA VALUE HAS A FORMAT ERROR.

Data obtained for a field violates its format specification such as non-numeric data in a numeric field.

(FOC211) THE PARENT SEGMENT NAME IS UNDEFINED.

A segment in the Master Dictionary refers to its parent by a name which has not been previously applied to another segment.

(FOC212) THE FILE DESCRIPTION CONTAINS AN ERROR.

Errors of various types have been discovered in a file description. The error messages have been displayed and this message summarizes the fact that it was a file in the Master Dictionary which has these errors. Hence, no further processing can occur for this file.

(FOC213) THE RECORD CONTAINS A FIELDNAME WHICH IS NOT RECOGNIZED.

The fieldname which has been referred to does not appear in the list of fieldnames or aliases for the file which is referenced.

(FOC214) THERE ARE MORE THAN 64 SEGMENTS DESCRIBED

The maximum number of segment names permitted in a file is 64.

(FOC215) FIRST SEGMENT SPECIFIED WITH NON-BLANK PARENT.

The first data segment described in the Dictionary refers to a parent segment. This is assumed to be an error.

- (FOC216) TERMINATOR OF \$ MISSING FROM END OF RECORD.
A record submitted in free format is missing a final terminator.
- (FOC217) SEGMENT NAME IS DUPLICATE AND NOT SAME AS PREV.
A segment with the same name has been encountered in a file definition. Segment names must be unique within a file.
- (FOC218) STRUCTURE HAS TOO MANY LEVELS.
The maximum number of levels in a hierarchy is 64.
- (FOC219) ERROR WRITING PAGE n OF FILE fn ft fm.
A disk writing error has occurred attempting to write page 'n' of file named fn ft fm. Backup the data file, erase it, and reload it onto the disk and retry procedure. Also check to see that the disk is attached with 'WRITE' privileges.
- (FOC220) ERROR READING PAGE n OF FILE fn ft fm.
A disk reading error has occurred attempting to read page 'n' of file named fn ft fm. Check the number of blocks associated with the physical file by issuing the CMS command LISTF fn ft fm (LABEL). The number of blocks must be a multiple of 5. If it is backup for the data file, erase it, release it and retry procedure.
- (FOC221) A HOLD FILE DOES NOT CURRENTLY EXIST.
A query about the fields in the HOLD file indicates that no HOLD file is currently active.
- (FOC222) THE WORD 'FILE' IS MISSING.
There is a syntax error. The word FILE is supposed to be provided.
- (FOC223) NON-NUMERICAL CHARACTER IN COMPUTATIONAL FIELD:
A numerical field is given a non-computational character.
- (FOC224) SYNTAX ERROR:
There is a syntax error serious enough to stop the interpretation.

(FOC225) ALPHANUMERIC FIELD HAS TOO MANY CHARACTERS:

The attribute for the FORMAT of the field has a smaller length.
The number of characters submitted exceeds this length.

(FOC226) FILEDEF MISSING FOR EXTERNAL FILE:

It is necessary to provide a FILEDEF so that the data may be located.

(FOC227) EXEC PROCEDURE NOT FOUND NAMED:

The stored procedure named cannot be located.

(FOC228) FORMAT ERROR:

The data values do not match the FORMAT attribute, i.e. characters in numeric field.

(FOC229) ERROR READING EXTERNAL DATA FILE:

An error occurred while reading the external data file. Check the FILEDEF, or other elements related to the presence of the external file.

(FOC230) MISSING ENDING QUOTE MARK:

A beginning quote is not balanced with an ending quote.

(FOC231) SEGTYPE ELEMENT IS NOT VALID...NOT S, KM, ETC.:

The attribute for SEGTYPE has an invalid value. Check the MASTER file description.

(FOC232)

(FOC233) TOO MANY FIELDS ARE INDEXED..LAST ONE IS:

The number of indexed fields plus the number of segments is too large.
Check the attribute FIELDTYPE on each field to verify that only selected fields have FILEDTYPE=I.

(FOC234) FIELDNAME USED AS KEY IS NOT FOUND INPATH TO THE SEGMENT:

The field named as the cross-reference key has not been encountered in the file description before the point where it is used as a cross-reference, hence is undefined at this point.

(FOC235) DESCRIPTION ERROR..SEGMENT WITH REAL DATA NOT FOUND FOR:

The segment to be retrieved via a cross-reference is not in the cross-reference file, or in the set of files in the dictionary.

(FOC236) LINKED FILE DOES NOT HAVE A MATCHING KEY FIELD NAMED:

The referencing, and cross-referenced file do not have the field named as the CRKEY in common. Check the names of the fields.

(FOC237) LINKED AND REAL FIELD HAVE DIFFERENT FORMATS:

The common CRKEY has a different FORMAT in the cross-referenced file, and the referencing file.

(FOC238) LINKED FIELD DOES NOT HAVE A FIELD-TYPE OF INDEX:

The cross-referenced field in the cross-referenced file does not have FILEDTYPE=I. Or, if the segment has been re-described in the referencing file this attribute is missing.

(FOC239) FORMAT DEFINITION IN ERROR...INTERNAL FIELD:

The format attribute is invalid. Check the attributes for the field.

(FOC240) DATA FILE MISSING LINKED SEGMENT NAMED:

The cross-referenced file does not have the segment named.

(FOC241) NO NAME PROVIDED FOR CROSS-REFERENCE FILE FOR:

The name of the cross-reference file is missing. Provide the name of the file in which the segment is described.

(FOC242) CAN'T FIND CROSS-REFERENCE FILE NAMED:

The cross-referenced file which is named cannot be located. File 'name' MASTER* is not found.

(FOC243) COMPUTATIONAL STATEMENT RECOGNIZED BUT NOT YET SUPPORTED.

The computational facility recognizes the stated operations but the FOCUS release does not yet perform the operation.

(FOC244) EXPRESSION IS TOO LONG .. BREAK IT UP INTO SMALLER PARTS.

The interpretation of the expression is stopped because the expression is too long. Break it up into two or more parts and re-submit.

(FOC245) COMPUTATIONAL BRANCHING ALONG MORE THAN ONE DATA PATH.

A conditional or unconditional branching statement was defined and the entire set of temporary defined fields for the file does not lie along a single top-to-bottom data path. (Permanently stored defined fields may still be referenced and need not lie along the common path of the temporarily defined fields).

(FOC246) COMPUTATIONAL STATEMENT REFERS TO MORE THAN ONE DATA PATH.

All data fields referenced by a computational expression must lie along a single top-to-bottom data path.

(FOC247) STORED BRANCHING STATEMENTS.

Conditional or unconditional branching statements may not be stored in the file description.

(FOC248) INCORRECT EVALUATION SEGMENT FOR PERMANENTLY STORED DEFINED FIELD.

The actual data references of a permanently defined field locate the expression at a segment other than the segment declared in the field dictionary.

(FOC249) UNDEFINED LABEL.

A conditional or unconditional branching statement refers to an undefined label.

(FOC250) DUPLICATE LABEL.

The label is associated with more than one computational expression.

(FOC251) COMPUTATIONAL EXPRESSION NOT RECOGNIZED.

The expression does not define a field, is not a conditional or unconditional branching statement, nor a label.

(FOC252) LABEL FORMAT ERROR.

A label must be 12 characters or less, and must not include blanks or special characters.

(FOC254) TEMPORARY FIELD REDEFINED WITH CHANGED FORMAT.

The same fieldname is used with two temporarily defined expressions, with different formats on the left-hand side. The second instance should either omit the format specification or repeat it exactly as before. (A temporarily defined field may redefine an actual field, with or without format change).

(FOC256) STRING OF DIGITS TOO LONG.

The expression contains a string of more than 16 consecutive digits without an intervening blank or special character. This exceeds the maximum size of a computational number.

(FOC257) MISSING QUOTE MARKS.

An open quote mark is not followed by a closing quote mark within 256 characters (including blanks).

(FOC258) FIELDNAME OR COMPUTATIONAL ELEMENT NOT RECOGNIZED.

An element in a computational expression is not recognized. It may be a mistyped fieldname or function.

(FOC259)

(FOC260) AN OPERATION IS MISSING AN ARGUMENT.

An arithmetic or logical operation is missing an argument - possibly two operations written in succession.

(FOC261) THE EXPRESSION IS INCOMPLETE BECAUSE AN OPERATION IS MISSING.

An arithmetic or logical operation is missing - possibly two fieldnames or constants written in succession.

(FOC262) UNBALANCED PARENTHESES.

The right and left parenthesis in the expression are not in balance.

(FOC263) EXTERNAL FUNCTION NOT FOUND.

An external function referred to in the expression could not be located within the accessible virtual disks - possibly a misspelled FOCUS function.

(FOC264) FUNCTION HAS INCORRECT NUMBER OF ARGUMENTS.

A FOCUS function was specified with too many or too few arguments.

(FOC265) OPERATION OR RELATION NOT RECOGNIZED.

An unidentified type of error is present in the statement, possibly a spelling or typing error.

(FOC266) IF..THEN..ELSE SYNTAX ERROR.

Missing or spurious IF, THEN, or ELSE clauses.

(FOC270) SYNTAX ERROR IN DECODE ELSE CLAUSE.

The ELSE clause in the decode is incorrect - it is not followed by a single decode element and a left parenthesis.

(FOC271) ODD NUMBER OF DECODE LIST ELEMENTS.

The decode list (including the ELSE clause) cannot be paired into code-decode elements or it is null.

(FOC272) FORMAT ERROR IN DECODE ELEMENT.

A code element does not conform to the format of the DECODE field or a decode element does not conform to the format of the left-hand side.

(FOC273) SYNTAX ERROR IN DECODE.

Missing parenthesis around decode list or list elements not separated by blanks or commas.

(FOC274)

(FOC275) FOCUS FUNCTION HAS INCORRECT NUMBER OF ARGUMENTS:

A function used in a calculation expression such as EDIT, does not have the correct number of arguments.

(FOC276) FIRST ARGUMENT OF 'EDIT' IS NUMERIC AND NOT A FILELDNAME:

The function EDIT requires either a fieldname as its first argument.

(FOC277) SECOND 'EDIT' ARGUMENT IS NOT ALPHANUMERIC:

When EDIT has a second argument it must be an alphanumeric mask. i.e. enclosed in single quotes e.g. '99\$\$/99'.

(FOC278) LAST ARGUMENT IN A USER FUNCTION IS NOT A FIELDNAME:

When a User function is called in a calculation expression the last argument must be a fieldname. The result of any calculation by the User Function is placed in this field.

(FOC279) NUMERIC ARGUMENTS IN PLACE WHERE ALPHA ARE CALLED FOR:

A function requires an alphanumeric value and it is given a numeric value. Check for missing quote marks but literal value.

(FOC280) COMPARISON BETWEEN COMPUTATIONAL AND ALPHA VALUES:

The expression is trying to compare alphanumeric values against numeric ones. Perhaps the single quote marks are missing. i.e. CVALUE EQ '756'.

(FOC281)

(FOC282) RESULT OF EXPRESSION NOT COMPATIBLE WITH FORMAT OF FIELD:

The expression either produces an alphanumeric result and the left-hand side is defined as numeric, or the reverse.

(FOC283)

(FOC284)

(FOC285) SUBSTITUTED LINE IN FOCEXEC TRUNCATED:

A line in the Dialogue Manager procedure is truncated when substitutions are made.

(FOC286) VARIABLE NAME EXCEEDS 12 CHARACTERS IN FOCEXEC:

A substitutable variable name must be 12 or less characters in length.

(FOC287) NUMBER OF EXEC VARIABLES EXCEEDS 256:

The maximum number of substitutable variables in a stored procedure has been exceeded. Reissue some of the names.

(FOC288) SUBSTITUTED VALUES EXCEED 4096 BYTES:

The length of all substitutable variables exceeds a limit. Reissue some of the variable names.

(FOC289) MISSING OR MISPLACED QUOTE IN REPLY, RETYPE COMPLETE REPLY.

The reply to a prompt has a misplaced quote so its length cannot be determined. Retype the line.

(FOC290) EQUAL SIGN MUST BE QUOTED IN A PROMPT REPLY, RETYPE COMPLETE REPLY.

A special character such as an equal sign must be enclosed in single quotes if it is part of the text. e.g. 'AB=BA'.

(FOC291) VALUE IN PROMPT REPLY EXCEEDS nnnn CHARS.

The reply to the prompt for the particular value is too many characters. The maximum length is shown in the message.

(FOC292) VALUE IN PROMPT REPLY IS NON-NUMERICAL:

The prompt reply expects a numeric response for the value being requested.

(FOC293) MISSING OR MISPLACED QUOTE IN EXEC ARGUMENT LINE:

The values in the EXEC line cannot be correctly interpreted because a quote is missing and one is therefore unbalanced.

(FOC294) SYNTAX ERROR IN EXEC ARGUMENT LINE:

The values supplies on the EXEC line cannot be interpreted. Check the positional, or set of variable equal value pairs.

(FOC295) VALUE MISSING FOR:

The EXEC procedure has encountered a substitutable variable for which no value has been provided. The procedure is not executed 'live' hence prompting cannot be used.

(FOC296) THE INTERPRETED EXEC EXCEEDS MAXIMUM NUMBER OF LINES.

A set of EXEC statements within an EXEC procedure requires too much temporary work space. Place some called procedures 'online'.

(FOC297) THE NUMBER OF FILES EXCEEDS MAXIMUM:

The number of files which must be simultaneously used to process a request exceeds 19.

(FOC298)

(FOC299)

(FOC300) MISPLACED QUOTE IN DATA REPLY, DISREGARD.

The reply to a prompt has a misplaced single quote. Hence quote marks are unbalanced and line can't be interpreted.

(FOC301) DATA CONTAINING EQUAL SIGN OR COMMA MUST BE QUOTED:

The data value contains a special character such as a comma or equal sign. The entire data value must be enclosed in single quotes. i.e. ADDRESS='NY,NY'.

(FOC302) ERROR WRITING FOCEXEC TEMPORARY FILE.

An error has been encountered using a temporary work file. Check if enough space exists.

(FOC303) CONTROL LINE NOT RECOGNIZED IN FOCEXEC:

A line in the stored procedure begins with a dash, but is not followed by a keyword, or valid label of 12 or less characters.

- (FOC304) LABEL INCORRECTLY SPECIFIED:
A GOTO statement does not refer to a valid name for a label. It cannot have special characters for example.
- (FOC305) SPECIFIED LABEL NOT FOUND:
A GOTO statement cannot find the label to branch to. Check the GOTO, or labels in the procedure.
- (FOC306) FILE ATTRIBUTE ELEMENT IS NOT RECOGNIZED:
The attribute used in a MASTER file description is not recognized.
- (FOC307) TOO MANY DATA FIELDS IN THE LINE:
A FREEFORM limited transaction has too many comma separators, hence refers to too many data values. The transaction is rejected as an error.
- (FOC308) ASSIGNED FIELD NOT MENTIONED IN TRANSACTION:
A field on a FREEFORM limited transaction is not on the assigned list. The transaction is rejected as an error.
- (FOC309) TRANSACTION INCOMPLETE:
A terminator has been encountered on a transaction before all of the fields on the FREEFORM list have been processed. The transaction is rejected as an error.
- (FOC310) PART OF REJECTED TRANSACTION:
A prior transaction has been rejected, and until a terminator has been encountered other lines are considered part of the rejected one.
- (FOC311) FIELD NOT CORRECTED FROM PREVIOUS TRANSACTION:
A previous transaction had an error in the data field referenced for which a new value has not yet been substituted.
- (FOC312) FIELD NOT SPECIFIED ON LHS OF ASSIGNMENT:
There is a syntax error because an equal sign has been encountered but no variable name before it.

(FOC313) PROMPT LINE CONTAINS NO &VARIABLE:

A -PROMPT subcommand line does not contain a variable name beginning with an '&' for which a value is to be supplied.

(FOC314)

(FOC315) INVALID VALUE ASSIGNED TO SYSTEM VARIABLE:

The system variable named is not assigned one of its valid values.

(FOC316) IF CONTROL STATEMENT BRANCHES TO INVALID LABEL:

The branching statement does not name a valid label as the point to be branched to. Possibly the label exceeds 12 characters, or contains a special non-allowed character.

(FOC317) MAXIMUM NUMBER OF POSITIONAL VARIABLES EXCEEDED:

The maximum number of numerical positional variables in the Dialogue Manager has been attained.

(FOC318) CANNOT SPECIFY FORMAT ON LHS OR SET:

In a -SET control statement the variable on the left-hand side of the equal sign cannot be assigned a format. The format in the Dialogue Manager is implied from the data which is provided.

(

(FOC401) SUBCOMMAND IS NOT RECOGNIZED:

The subcommand issued in a MODIFY procedure is not recognized. Check the list of subcommands.

(FOC402) UNRECOGNIZED WORD:

The syntax of the subcommand does not recognize a word. Check the syntax of the subcommand.

(FOC403) TRANSACTION DATA FILE CANNOT BE LOCATED:

The name of the file containing the transactions is provided on the DATA ON ddname subcommand. A FILEDEF may be missing for this name so it can't be located.

(FOC404) INCOMPLETE PROCEDURE:

The MODIFY procedure was not executed due to an error in the procedure, or termination by a QUIT or END by the operator before the end of the procedure. The message is a confirmation of the status of the procedure.

)

(FOC405) TRANS n REJECTED...DUPLICATE:

The MATCH conditions for transaction number n all match corresponding data base records. The option ON MATCH REJECT is in effect (explicitly or by default) and the transaction is rejected.

(FOC406) FIELDNAME IS NOT RECOGNIZED:

The name of the data field is not on the list of fields for this file. Check the spelling, or the list of fieldnames and aliases for the file.

(FOC407) TRUNCATED FIELDNAME IS NOT UNIQUE:

The short form used to identify a data field is not unique and could also apply to another field in the file. Use additional letters, or the full name to make it unique.

)

- (FOC408) INCORRECT OPTION AFTER 'ON':
The actions after the word 'ON' are only ON MATCH, or ON NOMATCH.
- (FOC409) CONFLICTING SUBCOMMANDS:
The subcommand conflicts with a previous one. They are mutually exclusive, i.e. when one or the other. For instance, ON MATCH DELETE conflicts with ON MATCH UPDATE.
- (FOC410) ACTIVITY AFTER WORD MATCH IS NOT RECOGNIZED:
The action used after MATCH is either UPDATE, DELETE, REJECT, or EXEC. Anything else is not recognized.
- (FOC411) ACTIVITY AFTER WORD NOMATCH IS NOT RECOGNIZED:
The action word after NOMATCH is either INCLUDE, or REJECT. Anything else is not recognized.
- (FOC412) FIELDNAME IS NOT A DEFINED VALIDITY TEST:
The name used in a VALIDATE subcommand does not have a DEFINE'd expression. Issue a ? DEFINE command for the current list of DEFINE'd names.
- (FOC413) NULL TRANSACTION AND/OR DUPLICATE TRANSACTION:
Two transactions are the same and the option ON MATCH REJECT is in effect explicitly or by default, or insufficient data appears on the transaction and the procedure hasn't given an action for its processing.
- (FOC414) NULL INCLUDE:
The transaction does not contain data for descendent segments although the procedure explicitly required them.
- (FOC415) TRANS n REJECTED...NOMATCH:
Transaction number n does not match corresponding data base fields as specified by the MATCH subcommand. The name of the segment (SEGNAME) on which no match occurred is displayed for reference, followed by the full transaction record.

- (FOC416) THE NUMBER OF FIELDS EXCEEDS THE NUMBER IN THE DICTIONARY:
Freeform data in which comma's delimit each field contain too many comma's. Or, a field is identified by its order number in the dictionary which exceeds the number of fields in the file.
- (FOC417) ADDITIONAL CORE IS NEEDED FOR RECORD WORK SPACE:
Before processing of transactions start work space is obtained. If This is not adequate then more core is needed. Add 12K to the amount of current storage and issue the CMS command DEFINE STORAGE AS N.
- (FOC418)
- (FOC419) FORM SUBCOMMAND ELEMENT or FIELDNAME NOT RECOGNIZED:
An element on the FORM subcommand such as the number of characters to process, cannot be identified.
- (FOC420) FIELD NAMED DOES NOT HAVE A COMPUTED EXPRESSION:
The name used on a COMPUTE subcommand does not appear in the list of computed items. Issue the ? DEFINE command to obtain the current list.
- (FOC421) TRANS n REJECTED... INVALID:
Transaction number n failed the VALIDATE test for it. The name of the test is displayed, followed by the complete transaction.
- (FOC422) FOCUS FILE IS NOT ON WRITABLE DISK, CAN'T MODIFY:
The MODIFY command can only be issued for FOCUS files which are on writable disks.
- (FOC423) ERROR WRITING LOG FILE NAMED:
An input/output error has occurred while attempting to write a transaction to the log file named. Possible error is an incorrect FILEDEF. Check the length parameter, LRECL, it must be large enough to permit a full 'logical' transaction to be written.
- (FOC424) TYPE OF LOG RECORD NOT RECOGNIZED:
The activity to LOG is not on the allowed list. e.g. NOMATCH, INVALID, DUPL, ERROR, TRANS, or ACCEPTS.

- (FOC425) SYNTAX ERROR ON LOG SUBCOMMAND:
A word is not recognized in the LOG SUBCOMMAND.
- (FOC426) LOG FILE MUST HAVE FILEDEF...PLEASE ISSUE IT FOR:
A FILEDEF for the particular LOG file has not been issued prior to the MODIFY procedure. LOG files must have FILEDEF's.
- (FOC427) LOG MESSAGES ARE EITHER ON OR OFF:
The syntax in the LOG subcommand mentions messages but is not either MSG ON or MSG OFF.
- (FOC428) DATA VALUE HAS INCORRECT FORMAT (EX:NOT NUMERIC):
On a free form transaction (comma delimited) a field value either exceeds the length specified for it, or a numeric field is given a non-numerical character. These are both format errors since they violate the FORMAT element for the field as stored in the MASTER dictionary.
- (FOC429) NESTED GROUPS ON FORM SUBCOMMAND, USE UNNESTED ONLY:
The FORM subcommand specifies repeating groups of fields which are enclosed between parenthesis. Within one set of parentheses another set has been detected, hence nesting the groups. This is not permitted.
- (FOC430) ATTEMPT TO INCLUDE ANOTHER SEGMENT IN UNIQUE CHAIN:
A segment has a SEGTYPE=U (unique). An attempt to include more than one segment on this chain has been detected. The transaction is treated as a duplicate and rejected. It is logged on the DUPLS LOG file if one has been specified. The duplicate acceptance option, ON MATCH INCLUDE, does not apply to unique segments.
- (FOC431)
- (FOC432) CANNOT MATCH ON, OR UPDATE DEFINED FIELD:
The list of fields to match on the MATCH subcommand must all exist in the data file, they cannot be temporary defined fields unless these have the same name as real fields.

(FOC433) START/STOP SUBCOMMAND REQUIRES INTEGER NUMBER:

The value following a START or STOP subcommand is not a positive integer number.

(FOC434) PROCEDURE CONTAINS CONFLICTING SUBCOMMANDS:

The MODIFY procedure contains instructions which are mutually exclusive, much as FIXFORM, and also FREEFORM.

(FOC435) MATCH CONDITIONS MISSING...SUPPLY THEM FIRST:

An ON MATCH, or ON NOMATCH subcommand precedes any MATCH condition. A line may have been left out of the procedure or inserted at the wrong point.

(FOC436) SOURCE OF DATA MISSING...NO DATA COMMAND:

The subcommand DATA or DATA ON ddname must be present in the procedure to identify the source of the transactions or start of data entry.

(FOC437) FORMAT ON FIXFORM INCOMPATIBLE WITH DATA FORMAT:

The type and/or length specifications on a FIXFORM element is not compatible with its FORMAT attribute. i.e. COST/P4 and COST is described as D8.2.

(FOC438) CANNOT MODIFY A FILE WHOSE FILESUFFIX IS NOT 'FOC'..?

Only MASTER file descriptions containing the attribute SUFFIX=FOC are considered FOCUS files. Add this value to the file description.