Burroughs Corporation

COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

PS 2219 0433

B1800/1700 DASDL

PRODUCT SPECIFICATION

REV LTR	REVISION ISSUE DATE	APPROVED BY	REVISIONS
B	1/26/79	J Hale	Updates to the Mark 8.0 Release
			1-6 <hex literals> and <file name> added
			1-9 - 1-15 New section on Multiprogramming, replacing Multiple Users.
			2-1 New syntax for DATA BASE
			2-2 KEYCOMPARE added to <option list>
			3-1 SECURITYTYPE and SECURITYUSE added to <audit trail>
			3-2 Added sentence pertaining to Attributes to item F. Added paragraph to item G pertaining to Attributes.
			4-1 - 4-2 Reference to "block" replaced with "table" Bridge (/1\) added to <control item> option
			4-4 New <item option> syntax
			4-5 Deleted sentence "Data sets and sets are not allowed" Added "The limit for the integer is 1023."
			4-6 Added <control item> syntax and discussion
			4-7 Added <variable format part> syntax and discussion
			4-8 Simple condition of the form data.name relation <literal> added Changed "essentially typeless" to "essentially of type alpha."
			4-9 Deleted "or hex literal." from item C.
			5-4 - 5-5 Syntax for <key> restructured; added syntax for <key name> NO DUPLICATES is the default for accesses.
			6-1 Added <data base name> to syntax for <physical attribute specification>
			6-2 Added TITLE, SECURITYTYPE and SECURITYUSE to <physical option> syntax. Syntax given for <database physical option>
			6-4 TITLE, SECURITYTYPE, SECURITYUSE and SECURITYGUARD attributes discussion added
			6-5 Deleted PRIME from STANDARD DATA SET. LOADFACTOR deleted from INDEX SEQUENTIAL ACCESS, SPLITFACTOR=75 added.
			7-1 STORAGE SPECIFICATION section replaced with SECURITYGUARD.
			8-1 - 8-14 Section 8 added; REMAPS and LOGICAL DATABASES Remaining sections renumbered
			9-1 <data set name> changed to <structure name> in syntax for PURGE statement
			10-2 FILE option now says See STRUCTURE.
			10-3 Discussion added to REORGANIZE and STRUCTURE options. UPDATE rewritten.
			10-4 - 10-7 CODE reqritten COBOLIB now COBOLIB <logical database name> RPG option and RPGLIB <logical database name> option added.
			11-1 GUARDFILE and RPB.LIB added.

TABLE OF CONTENTS

BASIC CONCEPTS

The purpose of this section is twofold:

a. To define the terms most frequently used in this document.

b. To describe the actions of DMSII in areas where the DASDL description interacts with other components of the system.

RELATED DOCUMENTATION

Name	Number
DMSII Audit and Recovery	P.S. 2219 0532
DMSII Reorganization	P.S. 2219 0540

DATA BASE DEFINITION

Basic Entities

A data base described in DASDL consists of data items, data sets, and sets. A data item is a field which contains varying data values. It is the smallest named container of information in the data base. Data items may be of various types such as alpha or packed decimal.

Related data items are organized into data records which reside in data sets. A data set is similar to a traditional data processing file, except that the data management system handles the mechanics of space allocation, record location and references (pointers) to other files. Any particular data record is said to be a member of the data set it is in.

A set is a collection of references to records in a data set. It provides a specialized access to that data set. It may, for example, provide efficient ordered access by an item in the data set; or it may select only certain members of the data set, based on a stated condition. More than one set may be applied to any given data set. Each set may include a reference to every member of the data set, or it may include references to only selected members. If it contains a reference to every member, it is called a spanning set; if it contains only selected members, it is called a subset.

All spanning sets are automatically maintained by the data management system. Whenever a record is added to or deleted from the data set, DMSII will perform a corresponding action for each spanning set. Subsets may be either automatic or manual. If the condition for inclusion is specified in DASDL, then the set is automatically maintained. If the condition is not specified, then the application programs must insert and remove entries in the set.

In DASDL, data sets and sets are both called structures. All structure names must be unique. Item names within a data set must be unique. However, items which are used as keys must be unique within the data base.

Embedded Structures

Many data relationships can be represented by a hierarchy, or tree-structure. In DASDL these organizations can be represented by including a data set among the items of a data record. If a data set does contain another data set as an item, then the contained data set is called an embedded data set, and the data record in which it is declared is called the owner or master of the embedded structure. Any number of embedded records may be under each master.

It is also possible to include sets as items in a data record. As with data sets, such items are called embedded. Most commonly, such sets apply to data sets within the same master, but they may also apply to other data sets.

If a structure is not embedded in any data set, then it is said to be disjoint.

The term "master" refers to a record of the data set in which a particular structure is embedded. Sometimes it is useful to talk about not only the master structure, but also the master's master, the master's master's master, and so forth. The term ancestor is used to refer to any of those structures, up to the disjoint master.

Manual References

Some applications cannot be represented as hierarchies. For such applications, it may be necessary for a data record to refer to another data record without regard to any tree-like structure. Such a reference might be to another record of the same data set, or to a record of another disjoint data set. It can be a member of a manual embedded subset. Manual references provide a network capability for describing data relationships. They are maintained by the application program using a special assignment syntax.

Access Characteristics Which Affect Programs

There are three logical classes of data retrieval, each represented by one or more DMSII structure types. In general, changing the type of structure within one access class will not affect application programs, but changing between classes may.

a. Unkeyed Access

In an unkeyed access, data records may not be retrieved in any special order. The data set structures which are in this class are standard and unordered; the sets are unordered lists. The application program should not infer any logical order from the results of "find next" on such an access; however, all records in the structure will be found exactly once.

b. Keyed Sequential Access

In sequential access, records may be retrieved in order by the key specified in DASDL. They may also be retrieved by specifying a particular value desired for that key. The set structures in this class are index sequential, ordered list, and access to ordered data sets. The application program may depend on the fact that a "find next" operation will locate the next record according to the sequencing specified in the key.

c. Keyed Random Access

In a random access, records may be retrieved by specifying a particular value desired. Serial access is not possible with this type of set structure. The set structure in this class is index random.

SYNTAX DIAGRAMS

The syntax presented in the figures and explanations that follow conform to railroad syntax as presented in software specifications released by Software Documentation at the Santa Barbara Plant.

The main line of development is from left to right, down on the left and up on the right. The beginning of the syntax is represented by one arrow (>) and is followed to its termination point (#). Continuation lines are indicated by double arrows (>>). Unless otherwise noted, optional entries are expected by the program in any order. Variables are presented in lower case, enclosed in brackets (<>). Upper case entries are required syntax. A bridge indicated by (/n\) shows the maximum number of times the line may be crossed. A bridge indicated by (/n*\) shows the minimum number of times that the line must be crossed.

ABBREVIATIONS

Variables may be abbreviated; the following abbreviations have been used in this document:

Variable	Abbreviation
attribute	attr
character	char
description	desc
identifier	id
integer	int
parameter	param
specification	spec
statement	stmt

Syntactic Entities

<identifier> — An identifier consists of uppercase alphanumeric characters and single embedded hyphens. The first character must be an alpha character. The maximum number of characters allowed is 17, and the identifier must not cross card boundaries.

Legal: ABC-2
XXX-XXX
YZ

Illegal: 3DE
XYZ-
-MN
abc

<literals> A literal can be either alpha (a string of characters), numeric (a string of digits), or Hex (a string of hexadecimal digits).

<alpha literals> Alpha literals must begin and end with a quote character. They may be continued on subsequent records. The first non-blank character of the continuation record must be a quote (") and is not considered to be a part of the alpha literal.

In order to prevent excessive blanks from being placed in the dictionary file, data base comments may be continued on subsequent records in the following manner. A quoted string is terminated by a quote on the current record. If all subsequent characters on the record are blank and the first non-blank character of the subsequent record is a quote, then the previous string is concatenated with the current string and stored in the dictionary file.

A quote may be placed in a quoted string by placing quotes adjacent to each other.

Input: "Here is how to put a "" in a string."
Produces: Here is how to put a " in a string.

A null string is represented by "".

If only blanks separate two strings, then they will be concatenated.

Input: "ABC" "DEF"
Produces: ABCDEF

<numeric literals> A numeric literal can be of two types. An unsigned integer consists of a string of digits not crossing a card boundary. An unsigned real is an unsigned integer with a

decimal point at the beginning, the end, or in the middle. It also cannot cross a card boundary. Exponential notation is not allowed.

Unsigned integer: 3894651

Unsigned real: .387
1.245
681.
1.0
0.9

<hex literals>

A hex literal must begin and end with an "@" character. All intervening characters must be hexadecimal digits; i.e., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. Hex literals are an alternate method of specifying a character string and may be used in the same manner as alpha literals.

A hex literal must have an even number of hex digits. A hex literal may not cross card boundaries.

Examples:

```
a.  ALPHA(7)  INITIALVALUE IS @130C0000@ "VAL";
b.  ALPHA(1)  INITIALVALUE IS @82@;
c.  ALPHA(3)  INITIALVALUE IS @F0@ @81@ @F2@;
```

<file name>

A file name may be of the form:

a. <multifile id>
b. <multifile id>/<file id>
c. <family name>/<multifile id>/
d. <family name>/<multifile id>/<file id>

The <family name>, <multifile id> and <file id> may be specified as identifiers or as alpha literals.

RECOVERABILITY

Types of Failure

A processing failure such as a Clear/Start may interrupt a logical update to the data base. For example, a data record may have been inserted into a data set but not into one of its automatic sets. To recover data base integrity after such a failure, it is necessary to preserve before images so that partial operations can be backed out.

A storage failure, on the other hand, implies that previously good data has deteriorated. Such failures are usually detected by hardware parity checks, though in some cases they are determined programmatically. Storage failures require a backup copy of the data area while it was still sound and a means to reconstruct the changes made since that dump. Normally, reconstruction is based on the audit trail maintained by the data management system, but ir some cases an installation may prefer to rerun its application programs.

Audit Trail

The data management system maintains an audit file whenever the audit option is set in DASDL. This file records "before" and "after" images of the data, in an abbreviated form when possible. No change is written to disk until the corresponding change is successfully written to the audit.

If a processing failure should occur, the "before" images are used to back out the partially completed operations. "After" images are used also, to update any good data which had not been written to disk at the time of the failure.

If a storage failure should occur, the after images are used to bring some previous copy of the data up-to-date. When applying after images, the data management system must coordinate the audit with the state of the data base at the time it was dumped.

For more detail on this subject see P.S. 2212 5470.

Transactions

A logical update to the data base may contain more than one primitive data management operation. For example, an update might consist of deleting a record from one data set and entering it in altered form into another. DMSII allows the application program to group such operations into a logical update or transaction. The data management recovery routines are transaction-oriented and always recover to a time when no partial transactions are reflected in the data base. A Clear/Start may interrupt a transaction; so may abnormal termination of a program which was in the middle of a transaction. In either case, if audit was specified in DASDL, the DMSII recovery routines will back up the data base to a "clean" point such that no partial transactions are reflected in the data base.

Restart Program Information

The application program must reprocess any transaction which was interrupted because of a processing failure such as a Clear/Start. To assist the application program in restarting, DMSII will record restart information of the programmer's choosing. If a failure occurs, the recovery routines will insure that the information corresponding to the last good transaction for which the restart area was audited is in a special data set called the restart data set. Using this information, the program can restart at the point to which the data base was recovered.

The program can detect that restart is necessary, if:

a. It discovers that a Clear/Start has occurred;

b. It discovers that another program caused some of its transactions to be aborted;

c. It is told by some manual method, as when reprocessing is needed after a storage failure.

MULTIPROGRAMMING

Locking versus Throughput

One of the most important design goals of DMSII is high throughput. To help achieve this goal, a high degree of parallelism is provided for in processing against the database. The locking strategy helps provide this parallelism.

It has been observed that a typical update operation against a database (i.e., a transaction) involves a relatively small number of operations and records. In order to provide for the integrity of this set of changes in a multiprogramming environment, some method of locking must be provided. Locking too many resources at one time decreases the possibility of parallelism and, therefore, decreases potential throughput. Locking too few resources results in unnecessary overhead and inconvenience to the user. DMSII balances this tradeoff by providing record level locking. Providing record level locking and allowing all user programs to access the database concurrently provides a good compromise between locking too much at once or too little.

Aborting Transactions

The only lock that persists for longer than the course of an individual database operation (e.g., STORE) is the lock a user may place on an individual record of a dataset. Specifically, index tables are not locked except during the course of a single database operation. Locking index tables (for example, until the end of a transaction) would provide the possibility of independence of transactions to the point that the last transaction of a single program could be backed out without affecting any other program, but the price paid in loss of potential parallelism is much too great.

Unlocking index tables as soon as possible, on the other hand, allows maximum parallelism, but causes all programs to be affected if the last transaction of any one of them must be backed out. When aborting a transaction, all transactions must be backed out until a point is reached at which no program was in the transaction state. This situation maximizes total throughput, because aborting transactions is an unusual case. The normal case is that no transactions are aborted. In general, maximum total throughput is achieved when the normal case is optimized, even at the expense of extra overhead in abnormal cases.

Because transactions in progress are allowed to complete before

the abort process commences, exception processing is minimized. The only verbs at which the abort exception need be handled by user programs are BEGINTRANSACTION, ENDTRANSACTION and CLOSE.

Deadlock

Record level locking introduces the possibility of deadlock. Consider the following sequence of events:

Program A locks record 1. Program B locks record 2. Program A tries to lock record 2, but since program B already has it locked, program A is blocked (temporarily suspended). Program B tries to lock record 1 (which program A already has locked). If program B is blocked at this point, the two programs are deadlocked or in a deadly embrace. They are in a circular wait, each waiting on the other. Deadlock is a circular waiting situation with any number of processes involved such that none of the processes can make any progress.

Besides deadlocks involving locked records, deadlocks can occur involving sync points and locked records. For example, program A enters transaction state (by executing BEGINTRANSACTION). Program B leaves transaction state (by executing ENDTRANSACTION), causing a syncpoint to be pending. Program B then locks record X and executes BEGINTRANSACTION. Since a syncpoint is pending, program B is blocked at BEGINTRANSACTION. Program A now tries to lock record X, which program B has locked. If program A is blocked, waiting for a record, the two programs will be deadlocked because the syncpoint that program B is waiting for cannot occur until program A finishes its transaction.

Internal to the access routines of DMSII, canonical locking orders and judicious use of the control state are used to avoid the problem of deadlock. If any pair of locks is always locked in (at most) one order and never in the reverse order, deadlock is impossible. It is impossible for DMSII to try to impose a canonical locking order for the user programs to use in locking records. The access routines detect deadlock just before it is about to occur. Instead, a process is selected from among those which would have been involved in the circular wait. That process unlocks all the records it has locked in the database. It then returns a deadlock exception to the user program.

The process chosen for the deadlock exception is chosen so that progress is guaranteed. Among those processes which would be involved in the circular wait, the one process with the lowest priority is chosen. As the result of a deadlock exception, a process is guaranteed that all its records are freed, but whether or not the process is in the transaction state is unaffected.

Only locked records and sync points are considered by the deadlock analysis of DMSII. All other sources of waits are ignored in the analysis, and it is the user's responsibility to see that deadlocks involving other resources don't occur. If any program in the transaction state is delayed (for any reason) long enough, it will eventually be time for a sync point, and from that time on, all new transactions will be blocked at BEGINTRANSACTION until the delayed program finishes its transaction.

When a program gets a deadlock exception, the access routines do not cause an abort, because the program may not yet have made any changes. In fact, when possible, the best way for user programs to be written is to lock all records involved in the transaction before making any changes. If any deadlock exceptions are encountered, handling them is a simple matter of branching back to the beginning of locking the records. If the records are locked before BEGINTRANSACTION is executed, then even a deadlock exception on BEGINTRANSACTION may be handled in this way.

Consistency

Every database has a set of consistency constraints. These are normally not written down anywhere, but their existence is real nevertheless. A phenomenon which is introduced by multiprogramming against a database is that transaction processing routines which preserve these consistency constraints when run alone no longer necessarily preserve them when run concurrently.

As a simple example, consider the consistency constraint that a data item A of record R1 must be equal to data item B of record R2. The following two transaction processing routines (in pseudo code) preserve this constraint when they are run alone.

```
LOCK R1                 LOCK R1
A := A * 2              A := A + 100
STORE R1                STORE R1
FREE R1                 FREE R1
LOCK R2                 LOCK R2
B := B * 2              B := B + 100
STORE R2                STORE R2
FREE R2                 FREE R2
```

However, when they are run concurrently, they may or may not preserve the consistency constraint that A=B. For example, the following execution order guarantees that A is unequal to B upon completion if A=B initially, no matter what the initial value of

A and B is.

```
LOCK R1
A := A * 2
STORE R1
FREE R1
```

```
                    STORE R1
                    FREE R1
                    LOCK R2
                    B := B + 100
                    STORE R2
                    FREE R2
```

```
LOCK R2
B := B * 2
STORE R2
FREE R2
```

It is the responsibility of the user programs to use the record locking capability of DMSII to maintain consistency in a multiprogramming environment. Before suggesting how this might be done, it is appropriate to discuss a problem which arises in the multiprogramming environment because of audit and recovery.

Reproducibility

The transactions which are backed out by DMSII recovery are normally reprocessed. However, in a multiprogramming environment, reproducing the same result obtained for these transactions before the recovery is a problem. This is especially true with an online database. Although batch processing against a database can often be organized such that each section of the database is updated in a single threaded mode, this is normally not feasible for an online database due to the inadequate throughput which would result.

As an example, consider an online banking database. Suppose a deposit to an account is made, and then shortly afterwards a withdrawal is made so large that the deposit is needed to cover it. After these transactions are processed, a recovery occurs, and these transactions along with others must be reprocessed. It would not be acceptable to cause an overdraft just because the transactions happened to be rerun in a different order.

Unless precautions are taken by the user application programs in what record locking conventions are followed, there may not be any way, even single threading the update in rerun mode, which reproduces the previous results (assuming exactly the same update

routines are used and that they do not make a distinction between normal mode and rerun mode). The user programs should use the record locking facilities of DMSII to control the manner in which concurrent transaction processing routines are interleaved.

Miscellaneous Locking Considerations

FIND does not cause the record retrieved to be locked. FIND should always be used with care, because it does not protect the results of any decisions made on the record retrieved.

LOCK (MODIFY) obtains the record exclusively as far as other LOCKers are concerned, but a FIND can retrieve the record, even if it is LOCKed.

When the record in the user work area is changed because of a FIND, LOCK, or DELETE, an implicit FREE on the previous current record is performed (if there was a previous current record and it was locked).

STORE does not free the record that was just changed or created. This aids in making a consistent set of changes to a collection of records during a transaction.

ENDTRANSACTION automatically frees all records the process has locked in the database involved. This helps make transactions independent and tends to help avoid starvation.

Programming Transaction Routines

A good programming strategy in a multiprogramming environment should achieve the following goals:

simplicity
efficiency
high total throughput
preservation of consistency constraints
reproducibility during rerun mode

Assume that it is never necessary to lock more than a small fixed number of records. Then the following strategy may be used.

Step 0: Obtain all necessary non-database input information needed for this transaction.

Step 1: LOCK all records involved in the transaction. This includes even those records which are retrieved but not altered. FIND is not used at all. If any deadlock

exceptions are encountered, branch to the beginning of this step.

Step 2: Execute BEGINTRANSACTION. If a deadlock exception is encountered, branch to the beginning of Step 1.

Step 3: All transactions at this point are independent of each other, because they have disjoint collections of records locked. Advantage can be taken of this fact for reproducibility. One possibility is to write the input data for this transaction to a global serial transaction history file. After a recovery, the backed out tansactions would be resubmitted from this file and processed in a single threaded fashion. The result of this reprocessing should reproduce the previous results of each transaction exactly. The transaction routines need not be aware of whether or not they are in rerun mode.

Step 4: Perform the database updates and execute ENDTRANSACTION. Do not free any records. Let ENDTRANSACTION free all locked records.

The amount of processing done in the transaction state should be minimized in order to avoid excessive synchronizatcn delay at sync point time. Special care shculd be taken to avoid potentially long waits, such as file opens, waiting for datacomm input, etc., in the transaction state. One advantage of the above strategy is the elimination of potentially long waits on locked records in the transaction state.

One possible problem with the above strategy is the existence of bottleneck records, i.e., records that must be locked by many different transactions, because they contain totals which must be updated, etc. The problem with such records is that in order to avoid a locking bottleneck they must be kept locked for as short a time as possible.

The above strategy may be modified for such records as follows:

Instead of locking all records before BEGINTRANSACTION, lock all except the bottleneck records. Define a canonical locking order such that they may be locked after BEGINTRANSACTION is executed without causing any deadlocks. The bottleneck records are all locked in transaction state, updated, and immediately freed. The point of independence of concurrent transactions is then between the LOCK and the FREE of the innermost bottleneck record.

In the modified strategy as well as the original strategy, FIND is not used, only LOCK. Furthermore, in both strategies, once a

record has been freed, no more records are locked, and the point of independence of concurrent transactions is anywhere between the last lock (or BEGINTRANSACTION) and the first FREE. Thus, both strategies may be divided into two phases. During the first phase, records may be locked but not freed, and during the second phase records may be freed but not locked, and the point of independence is anywhere between the two phases.

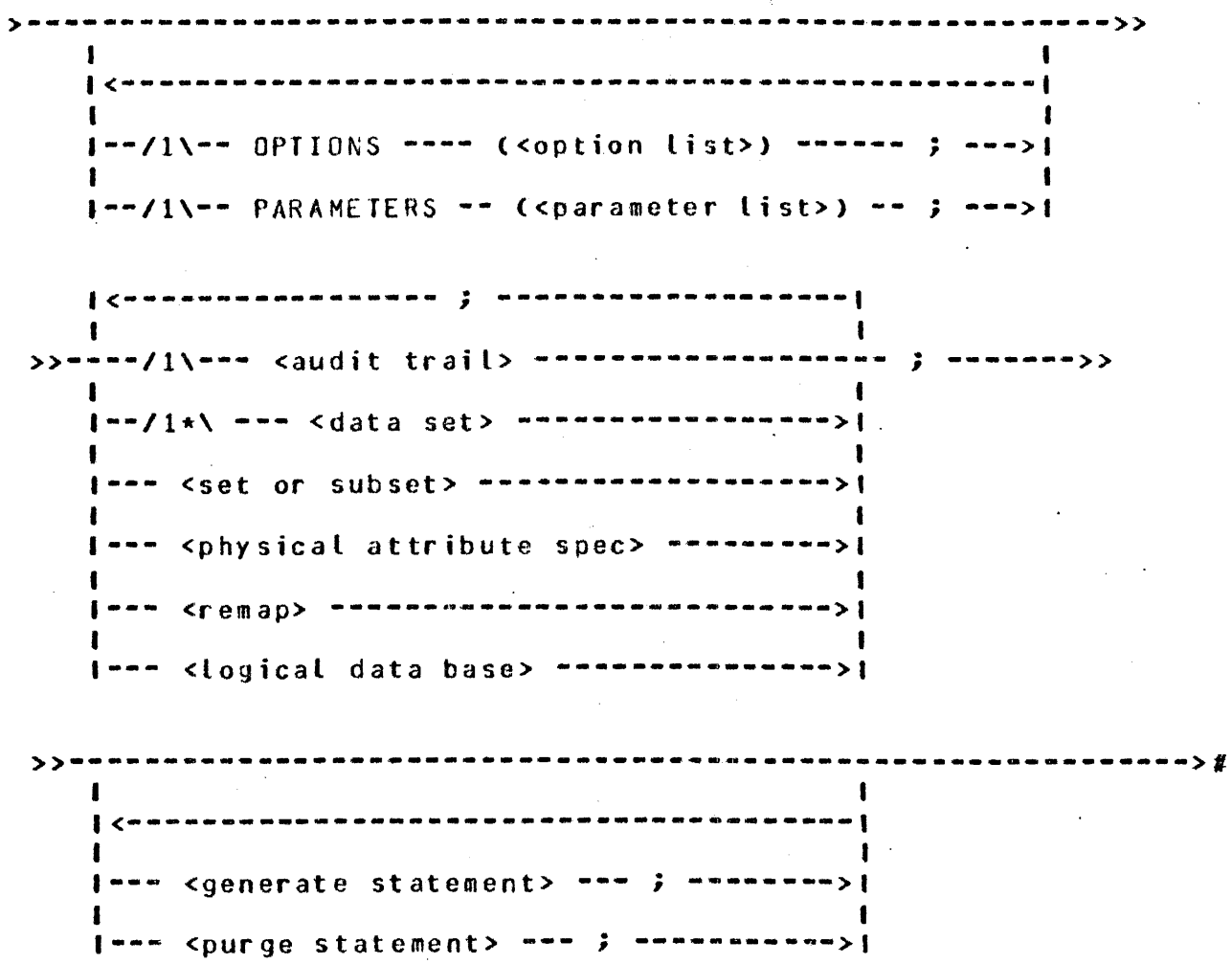
This two phase technique has several important properties. If all transactions are two phase but otherwise arbitrary, then any consistency constraints preserved by the routines when running alone are also preserved no matter what mix of them is run concurrently. Furthermore, no individual transaction routine sees any of the consistency constraints violated except as it temporarily violates them itself during the course of its update. Finally, the point of independence of concurrent transactions between the two phases provides a handle on the problem of reproducibility, because a serial schedule can be created which will reproduce the results of the concurrent updates for every transaction.

The two phase requirement is not strong enough, however, in the case that it is noticed that a certain transaction causes an abort recovery every time it is submitted (due, perhaps, to a programming error). In this case, to make any progress, the transaction in error must be erased from the global transaction history file and not resubmitted. If some of the results of the transactions have been displayed on terminals, then there are two alternatives to insure that the results of the rerun are the same as what has already been displayed. The first is to not unlock any records in the transaction, but to let ENDTRANSACTION do it instead. This insures that no transaction can depend upon another transaction's changes until the other transaction has executed ENDTRANSACTION, and will not, therefore, be erased from the transaction history file for causing an abort recovery. The other alternative is to execute ENDTRANSACTION SYNC to protect against aborts, but this will probably cause unacceptable throughput degradation due to excessive synchronization overhead.

The two phase technique must be modified somewhat when a large number of records must be retrieved during a transaction, because DMSII does not provide a convenient way for a program to keep a large number of records locked. Programmatic conventions must be defined such that in order to access a certain class of records, a specified record must first be locked. This technique single threads access to the defined class of records. The records in such a class are exempt from the two phase requirement, but the control record is not. An example of this technique is for the user to create special control records which are used only for locking purposes.

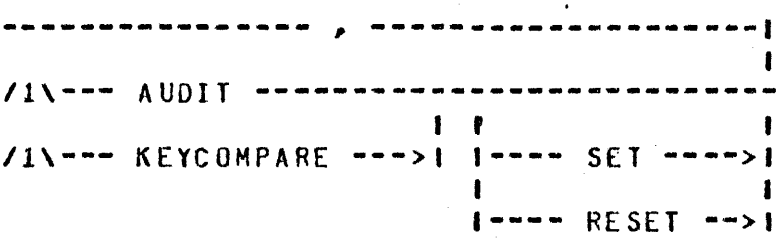
DATA BASE

Syntax: <data base>



DATA BASE OPTIONS

Syntax: <option list>



Semantics:

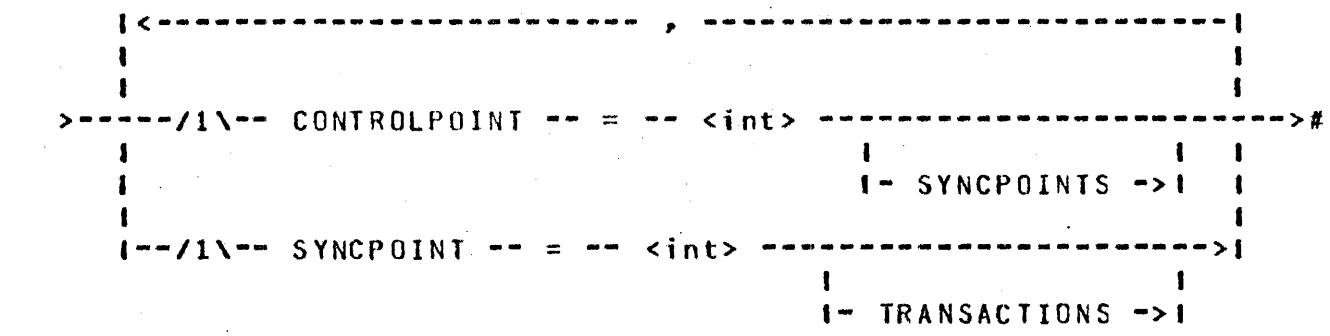
Options are used to determine the inclusion of major features of the DMSII procedures:

A. The AUDIT option allows auditing of the data base. When this option is used, the data base must include exactly one restart data set, described later. If set, the DMSII procedures will maintain an audit trail of changes to the data base so that data base integrity can be restored in the case of processing failures or storage failures. If neither SET nor RESET is specified, SET is assumed.

B. The KEYCOMPARE option allows the system to verify the key of a retrieved data record against the key of retrieval. If neither SET nor RESET is specified, SET is assumed.

DATA BASE PARAMETERS

Syntax: <parameter list>



Semantics:

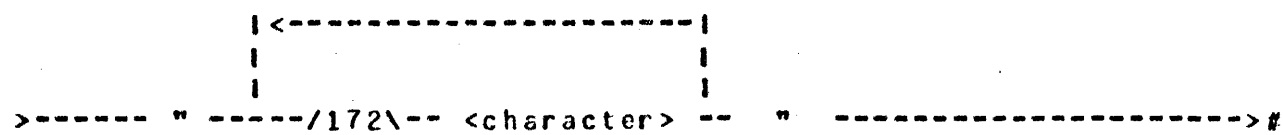
The <parameter list> is used to provide certain run-time parameters to DMSII:

A. CONTROLPOINT specifies how many syncpoints should occur between controlpoints. Default is 10 syncpoints. CONTROLPOINT applies only to audited data bases.

B. SYNCPOINT specifies how many transactions should occur between syncpoints. Default is 5 transactions. SYNCPOINT applies only to audited data bases.

COMMENT

Syntax: <comment>

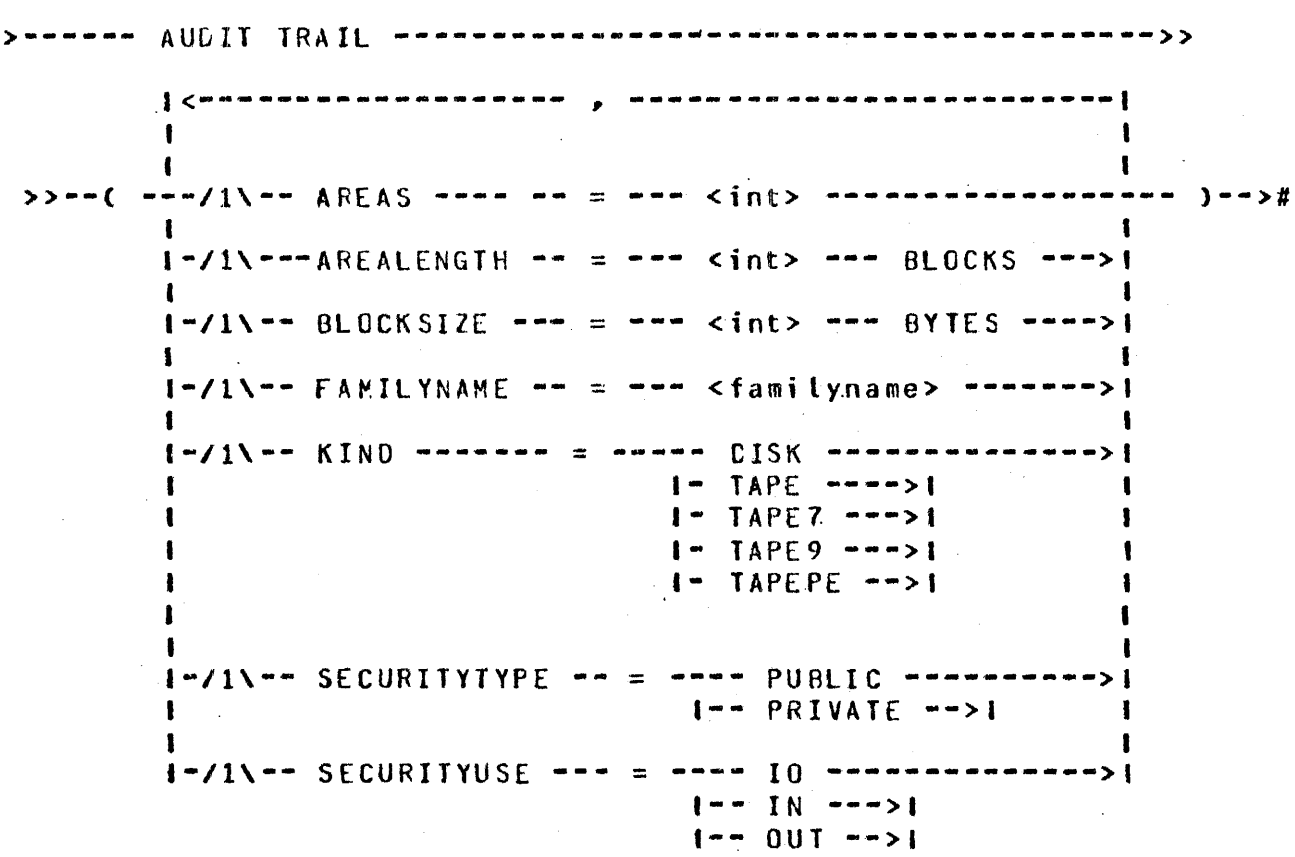


Semantics:

The quoted comment (limit = 172 characters) provides a facility for including descriptive information in the data base description tables.

AUDII

Syntax: <audit trail>



Semantics:

Defaults for audit trail attributes:

A. AREAS is 20.

B. AREALENGTH is 100 blocks

C. BLOCKSIZE is 1800 bytes.

D. KIND is DISK.

E. FAMILYNAME is SYSTEM DISK

F. The attribute SECURITYTYPE specifies who, apart from the owner, may access this file. The mnemonics of the SECURITYTYPE attributes are as follows:

PRIVATE: Implies that only a privileged user or the owner may access the file.

PUBLIC: Allows access to any user who references the file.

The default value for SECURITYTYPE is PUBLIC if the multifile-id of the file does not contain a usercode. Otherwise, the security attribute of the usercode is used. This attribute is applicable only if kind=DISK, and access is not through DMSII.

G. The attribute SECURITYUSE specifies the manner in which a disk file that is protected by security may be accessed. The default value for SECURITYUSE is IO. The mnemonics of the SECURITYUSE attribute are as follows:

IN: Indicates that read-only access is allowed to the file.

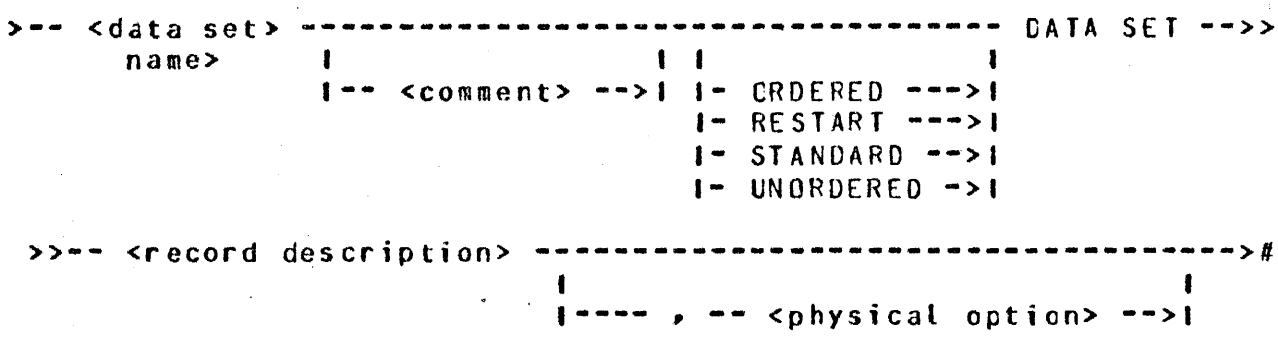
OUT: Indicates that write-only access is allowed to the file.

IO: Indicates that read/write access is allowed to the file.

This attribute is applicable only if kind=DISK, and access is not through DMSII.

DATA SETS

Syntax: <data set>



Semantics:

A <data set> description provides for specification of both the physical and logical structure of a file. The different data set structures are:

A. CRDERED

This structure requires exactly one <access> to define the key. Records in each table of an ordered data set are kept in physical order based on the key. Tables are linked together (also preserving the key order). All records in a table belong to the same master. Only the one access method is permitted. No sets may be associated with such a data set. This structure must be embedded.

B. RESTART

The RESTART data set is a special type of STANDARD data set used for audit and recovery. Exactly one data set with its description must be designated as the RESTART data set if the AUDIT option is specified. This structure may not be an embedded data set, and may have NO embedded structures within it.

C. STANDARD

Records will be placed in a system selected location. This structure may not be embedded.

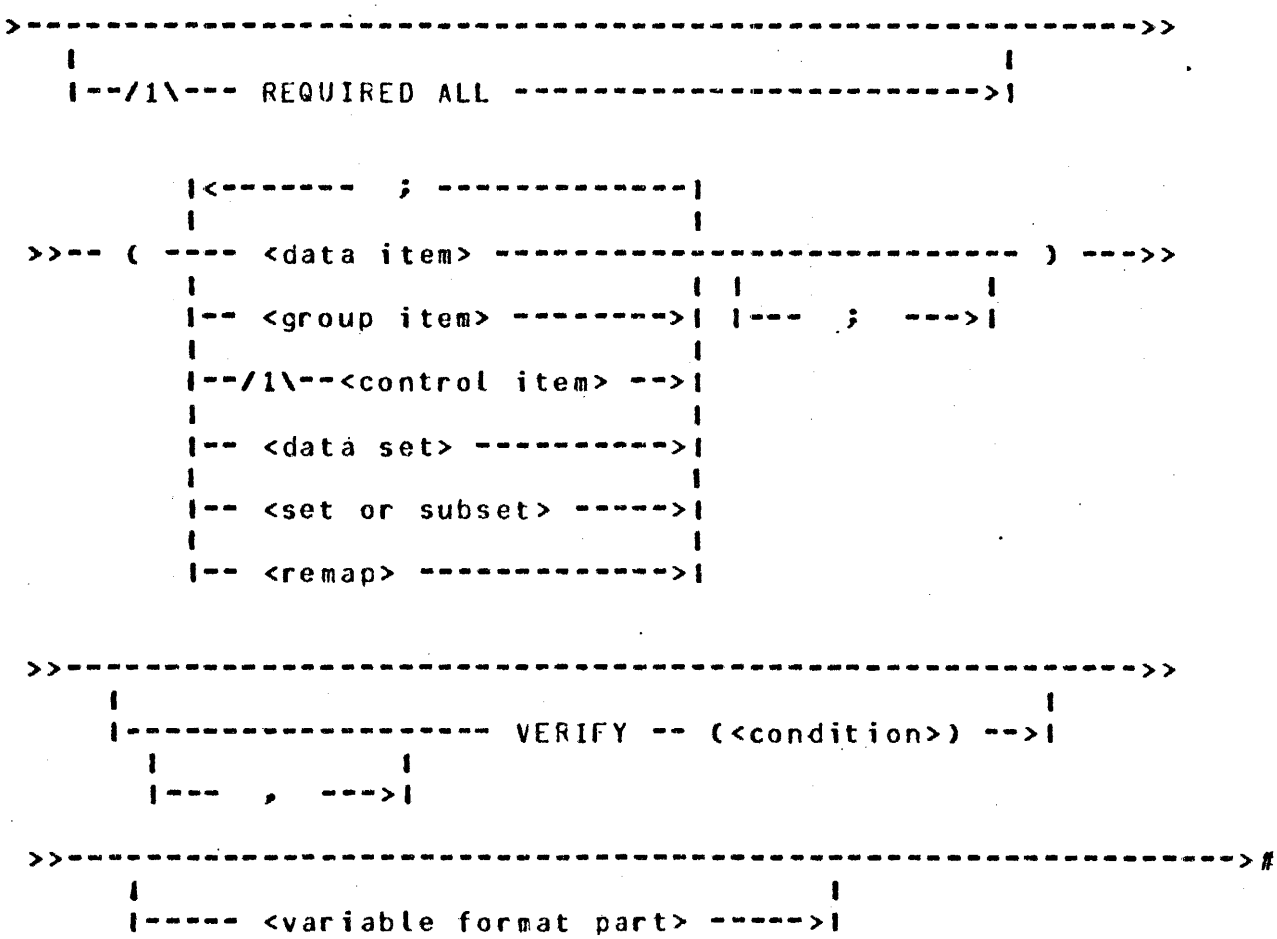
D. UNORDERED

Records will be placed in a system selected location. Physical tables are maintained in a linked list. This structure must be embedded. All records in a table belong to the same owner. No sets or accesses may be associated with such a data set.

The default data set structure is STANDARD.

DATA SET RECORD

Syntax: <record description>



Semantics:

A <record description> specifies the format of a record of the data set.

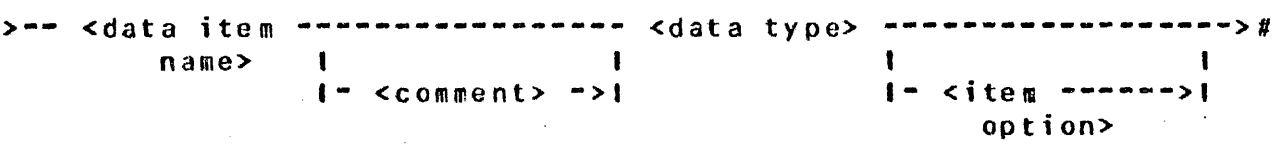
The physical records of the structure associated with the data set are referred to as members of that data set.

REQUIRED ALL is equivalent to specifying REQUIRED on each <data item> of the data record.

The VERIFY clause specifies a condition to be satisfied by items in the potential record of a data set. If the condition is not satisfied, then the record will not be stored, and an exception returned.

Data Item

Syntax: <data item>

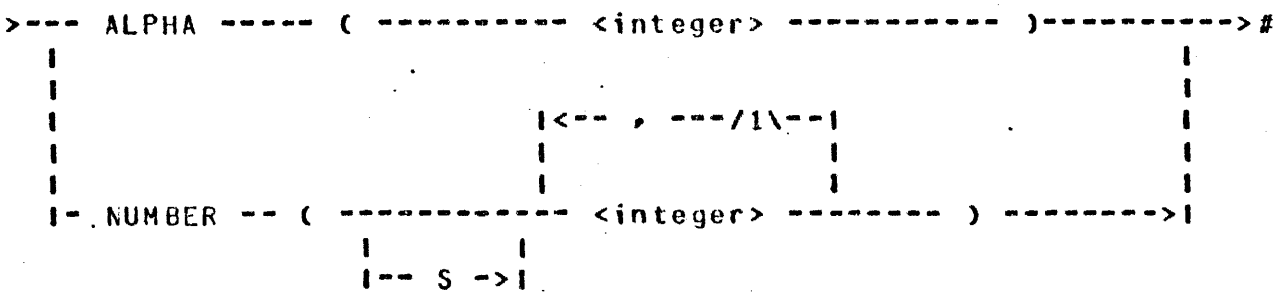


Semantics:

A <data item> is a field in a record whose value is controlled by application programs subject to the restrictions placed on data values by the <item option> clause and the VERIFY clause.

Data Type

Syntax: <data type>



Semantics:

ALPHA items are stored as EBCDIC characters. The NULL value is all bits on. The maximum size of an ALPHA item is 8191 characters. NUMBER items are stored as 4-bit digits (i.e., packed decimal) with a maximum size including sign digit of 23 digits (12 if used as a key item). The null value is all bits on.

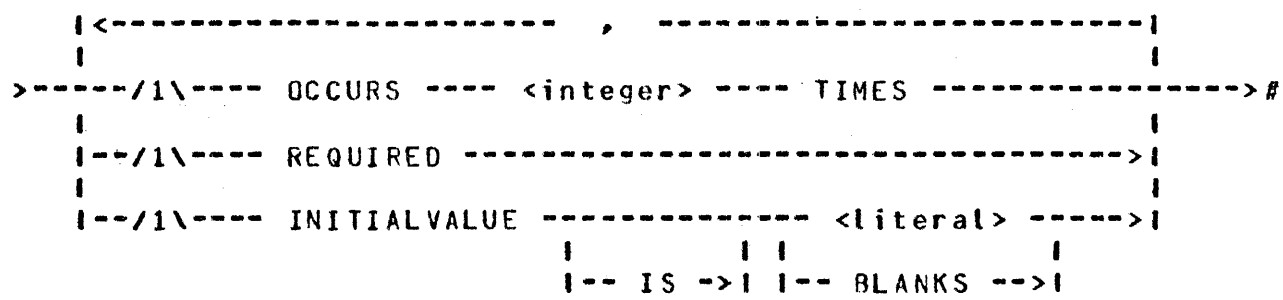
NUMBER declarations are treated as follows:

```
X NUMBER (S5,2)
```

"X" occupies one digit for the sign followed by five digits for the number and the decimal point is after the third digit.

Item Option

Syntax: <item option>



Semantics:

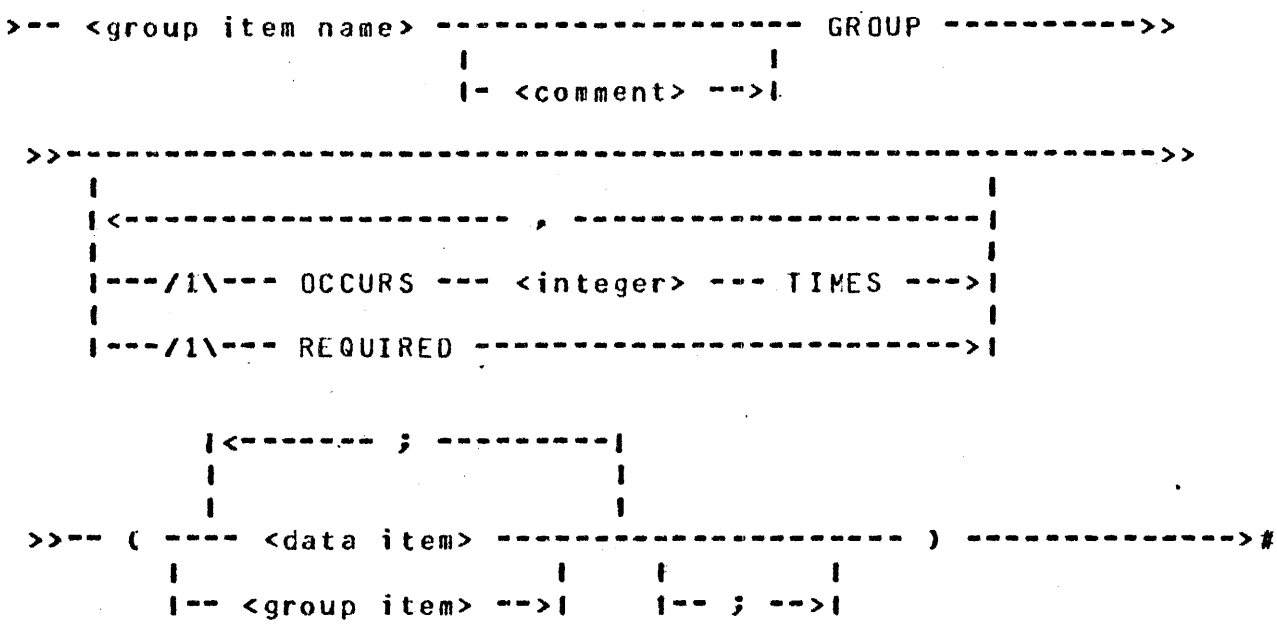
REQUIRED items must be present and not NULL at store-time. The REQUIRED ALL option for a data set will make all items REQUIRED.

INITIALVALUE specifies a means of setting item values at the time of creation of a record. Items with INITALVALUE specified will be initialized to that value on CREATE. If the size of the literal is less than that of the item, the literal will be stored into the item in accordance with COBOL conventions. If the size of the literal is larger than that of the item, then an error will be given by DASDL. Items with no INITIALVALUE specified will be initialized to null (all ones).

OCCURS causes the item to be repeated the specified number of times. The limit for the <integer> is 1023.

Group Item

Syntax: <group item>



Semantics:

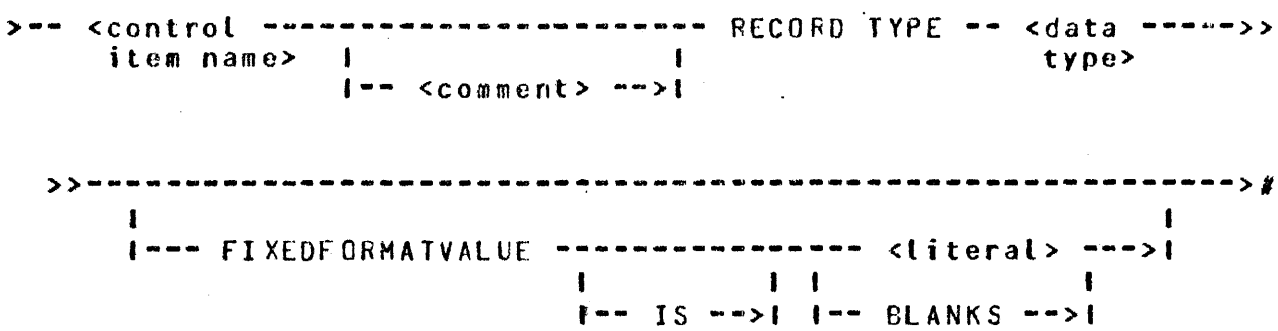
<group item>s are alpha items that themselves contain items. Items within a group are declared at a level one greater than the level of the group.

Items that may belong to groups are restricted to data items and further group items. <group item>s may be REQUIRED or may occur a specific number of times. If a <group item> is REQUIRED, each item in the group is required. group is required.

An OCCURS group may contain further OCCURS <group item>s or OCCURS <data item>s. However, a maximum of three levels of OCCURS is allowed. The limit for the <integer> is 1023.

Control Item

Syntax: <control item>



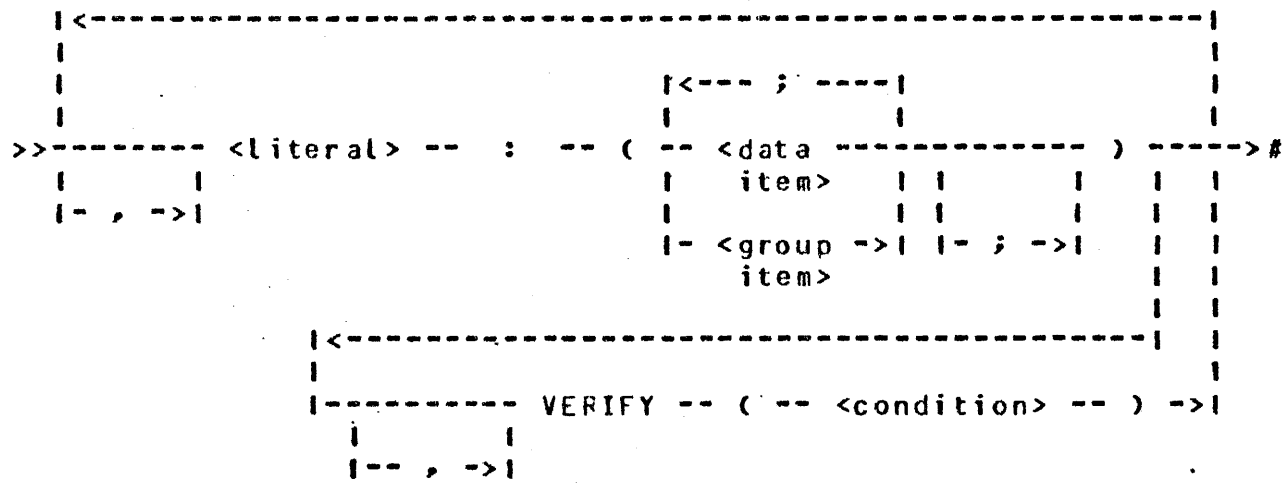
Semantics:

A <control item> is maintained in the record in which it is described and it has specific system meaning.

RECORD TYPE is required for variable format records. Its value determines which format a particular record has. Each data set has a maximum of one RECORD TYPE control item. The FIXEDFORMATVALUE clause specifies the value that the RECORD TYPE field will have when the record consists of the fixed portion only. If a FIXEDFORMATVALUE is not specified and the data type of the RECORD TYPE is ALPHA, a value of BLANKS is assumed. If the data type is NUMBER, then a value of zero is assumed. RECORD TYPE is a read only field. An attempt to modify this field will result in a data error.

Variable Format Part

Syntax: <variable format part>



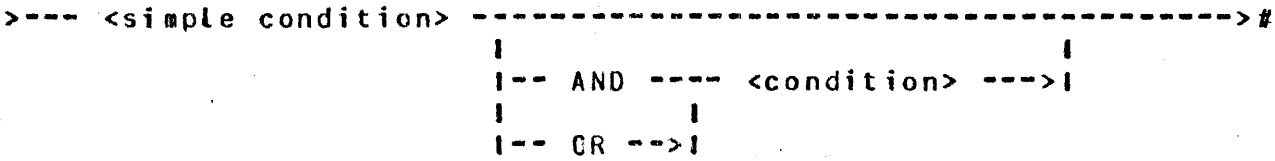
Semantics:

A RECORD TYPE control item must be declared in the fixed part of the record. The literal specified in the <variable format part> description must be of the same type as the RECORD TYPE control item. Each literal must be unique.

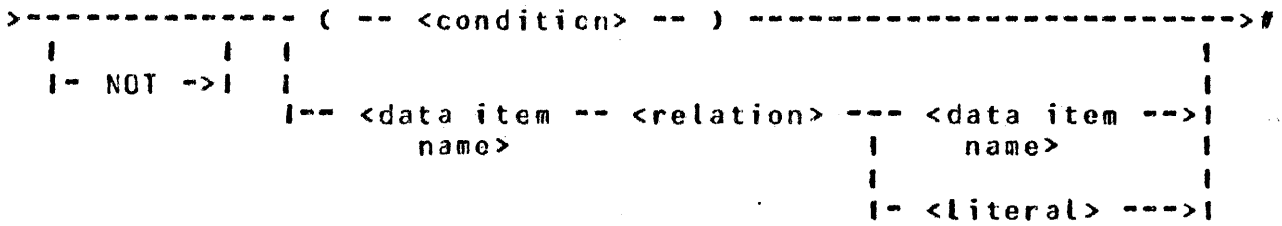
Each different format is constructed by taking the fixed portion of the record and appending the appropriate variable portion. The variable portion is selected by comparing the RECORD TYPE item with its possible values as specified by the literal fields in the <variable format part> specification. If the RECORD TYPE item is equal to the FIXEDFORMATVALUE, the record consists solely of the fixed portion.

CONDITION

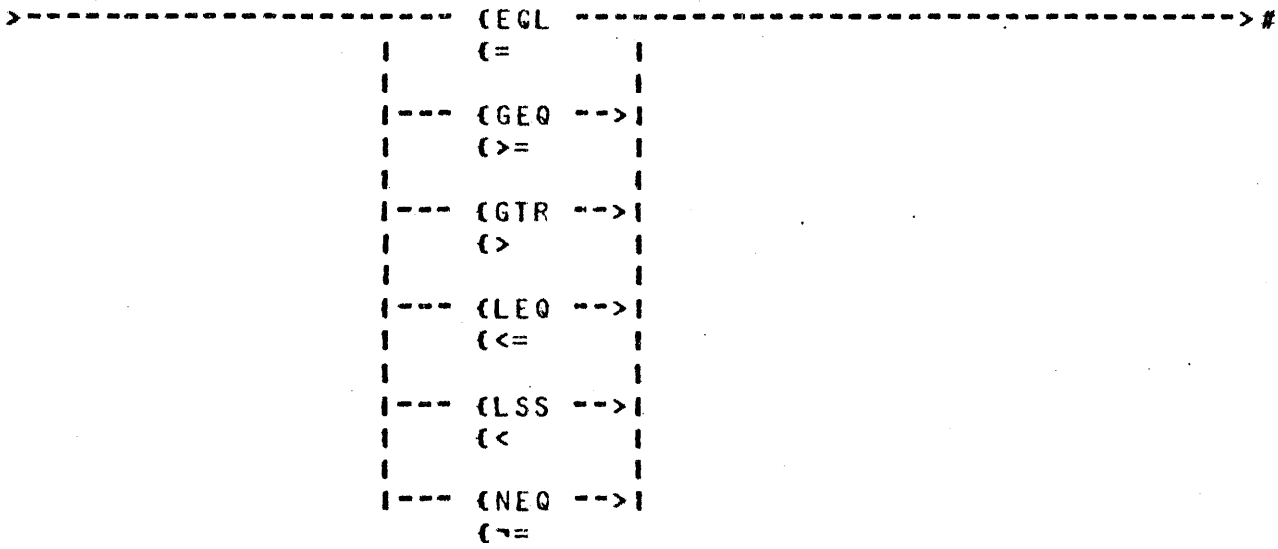
Syntax: <condition>



Syntax: <simple condition>



Syntax: <relation>



Semantics:

In any simple condition of the form:

--- <data.name.1> --- <relation> --- <data.name.2> -->

the data items 1 and 2 must be of similar type (e.g., ALPHA-ALPHA, NUMBER-NUMBER).

In any simple condition of the form:

--- <data.name> ----- <relation> --- <literal> ------>

the data item and the literal must be of similar type, except in the case of a hex literal which is essentially of type alpha. The legal combinations are:

a. ALPHA - alpha literal or hex literal
b. GROUP - alpha literal or hex literal
c. NUMBER - numeric literal

In cases of alphanumeric compares where <data.name.1> and <data.name.2> or <literal> are of different length, the comparison is made after extending the shorter string according to COBOL rules.

Where <data.name.1> or <data.name.2> is defined within the scope of an OCCURS clause, all necessary subscripts must be specified.

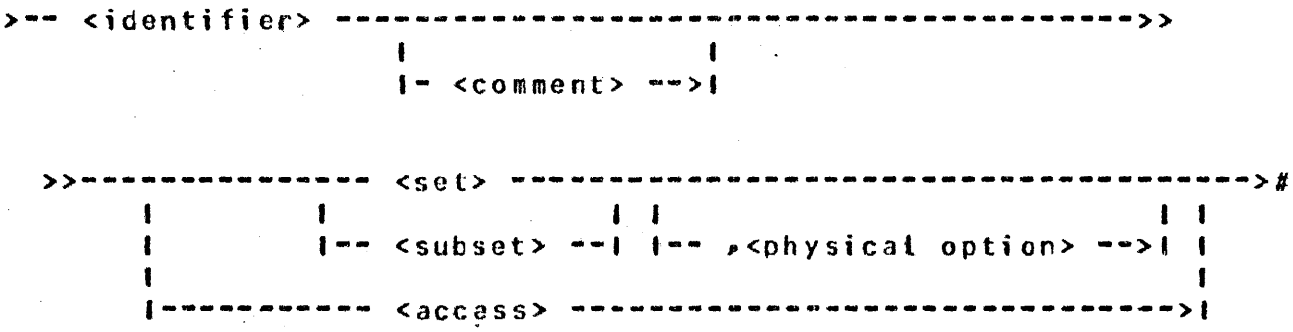
Precedence of operator, from highest to lowest, is:

NOT
AND
OR

Evaluation is from left to right with due respect for parentheses.

SETS AND ACCESSES

Syntax: <set or subset>



Semantics:

A <set or subset> description provides logical and physical specifications of an access method which will be used in storage and retrieval of data.

A <set> includes one reference for each and every record in the associated data set.

A <subset> includes, in general, references to only some of the records in the associated data set.

An <access> functions similarly to a set, but may be applied only to ORDERED data sets. Exactly one <access> must be declared for each such data set. No physical tables are associated with <access>s.

SETS AND SUBSETS

Syntax: <set>

Syntax: <subset>

Semantics:

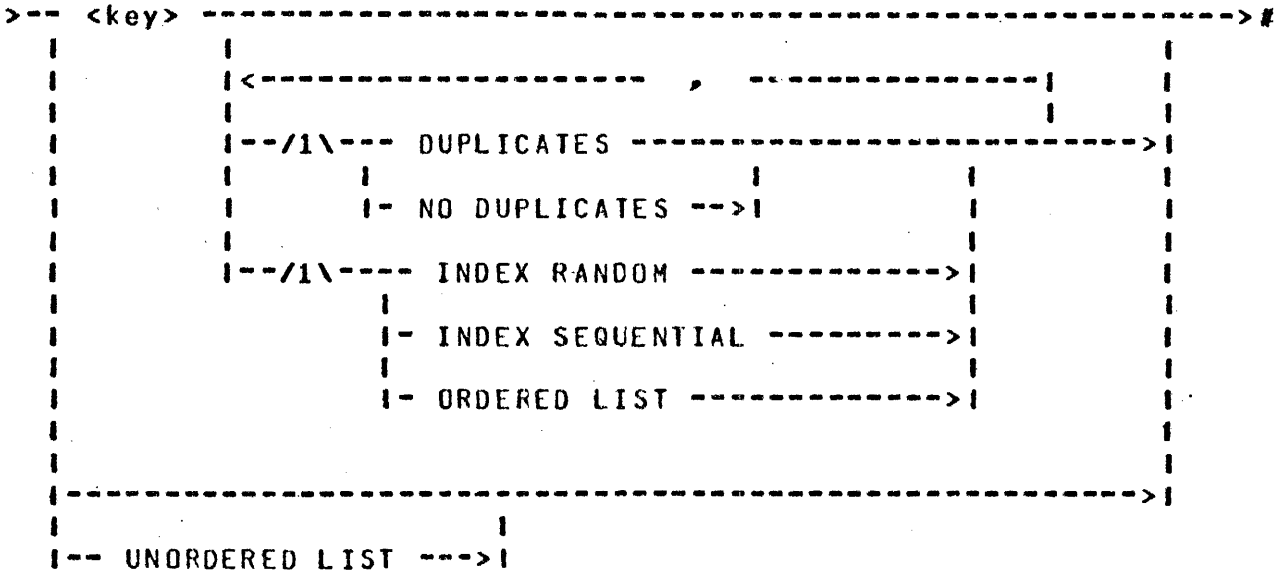
<set> may only reference data sets declared at the same level. The set and data set must both be disjoint.

A <subset> which specifies a WHERE <condition> is called an automatic subset. A <subset> which does not specify a WHERE <condition> is called a manual subset. An automatic subset includes one reference for each record in the associated data set which satisfies the <condition>. A manual subset ircludes only those references explicitly inserted by application programs. Automatic subsets may only reference data sets at the same level. Automatic subsets must be disjoint.

A manual subset may reference any standard data set which is disjoint. A manual subset must be embedded.

Key Structure

Syntax: <key structure>



Semantics:

A <key structure> defines the organization of a set or subset. It may include a key on which records may be retrieved. In addition, it may specify the table structures (e.g., INDEX SEQUENTIAL).

The access techniques available are:

A. ORDERED LIST

An ordered list is a collection of tables. Entries within the table are ordered on key.

Each table entry consists of a key and an address.

Tables are linked together.

This structure may only be used for manual subsets with a <key>.

B. INDEX SEQUENTIAL

Coarse table entries point to fine tables or other coarse tables. Fine table entries point to data records in the associated data set.

Entries within both tables are in sequence on key.

When a table becomes full, the table is split into two tables, based on the SPLITFACTOR. Refer to <physical option> under Physical Attribute Specification.

Table entries consist of a key and an address.

This structure may only be used for automatic subsets and sets.

C. INDEX RANDOM

MODULUS represents the number of basic tables in the set. Refer to <physical option> under Physical Attribute Specification.

A hashing algorithm is applied to the symbolic key to obtain a basic table number in which to locate the table entry. The table entry consists of the key and address of the data record. If the selected basic table is full, an overflow table will be used.

This structure may only be used for sets.

D. UNORDERED LIST

An unordered list is a collection of tables.

Table entries consist of addresses only.

The tables are linked together.

This structure may only be used on manual subsets without a <key>. If only one entry has been inserted into the manual subset, then that entry is carried in the parent record (no list tables are allocated).

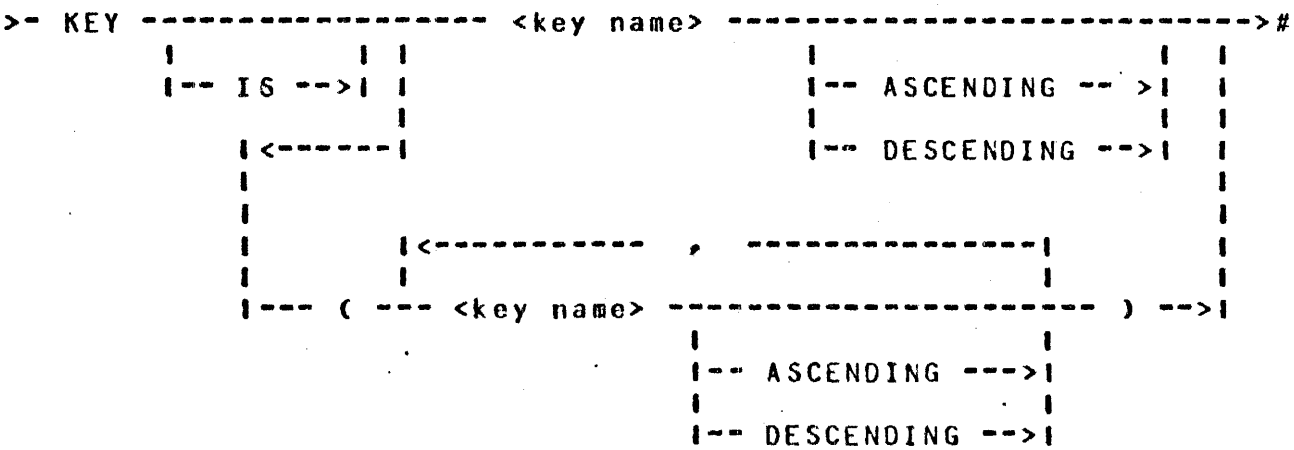
The ordering of duplicate key records is unspecified.

Default organization for sets with keys is INDEX-SEQUENTIAL for disjoint sets or automatic subsets, and ORDERED LIST for manual subsets with keys. Default organization for manual subsets without keys is UNORDERED LIST.

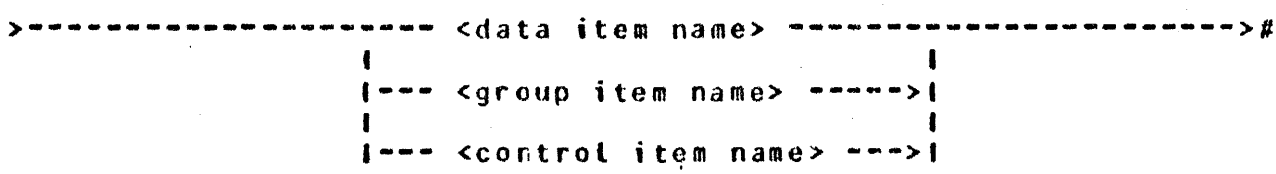
NO DUPLICATES is the default for all sets or subsets with keys. DUPLICATES must be specified on any automatic subset with keys.

Key

Syntax: <key>



Syntax: <key name>



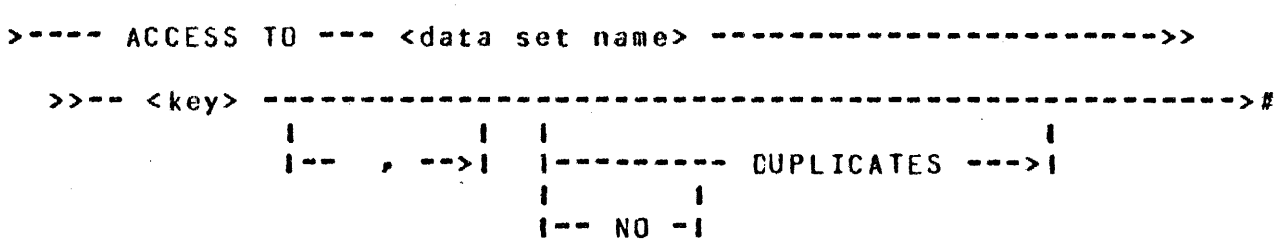
Semantics:

Each key name in a key must refer to an item in the data set on which the set or subset is declared. The default ordering of records is keys ascending. Subscripted items may not be used as keys. Numeric data items in a key are restricted to 12 digits in length. Key items are implicitly required.

Index Random sets may not have descending key items.

ACCESSES

Syntax: <access>



Semantics:

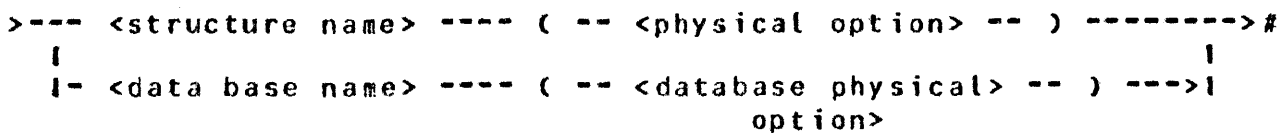
One, and only one, <access> must be declared for each ORDERED data set.

Accesses may not be declared for any type of dataset except ORDERED.

NO DUPLICATES is the default for accesses.

PHYSICAL ATTRIBUTE SPECIFICATION

Syntax: <physical attribute specification>



Semantics:

<physical attribute specification>s are used to specify the physical structure of the data base.

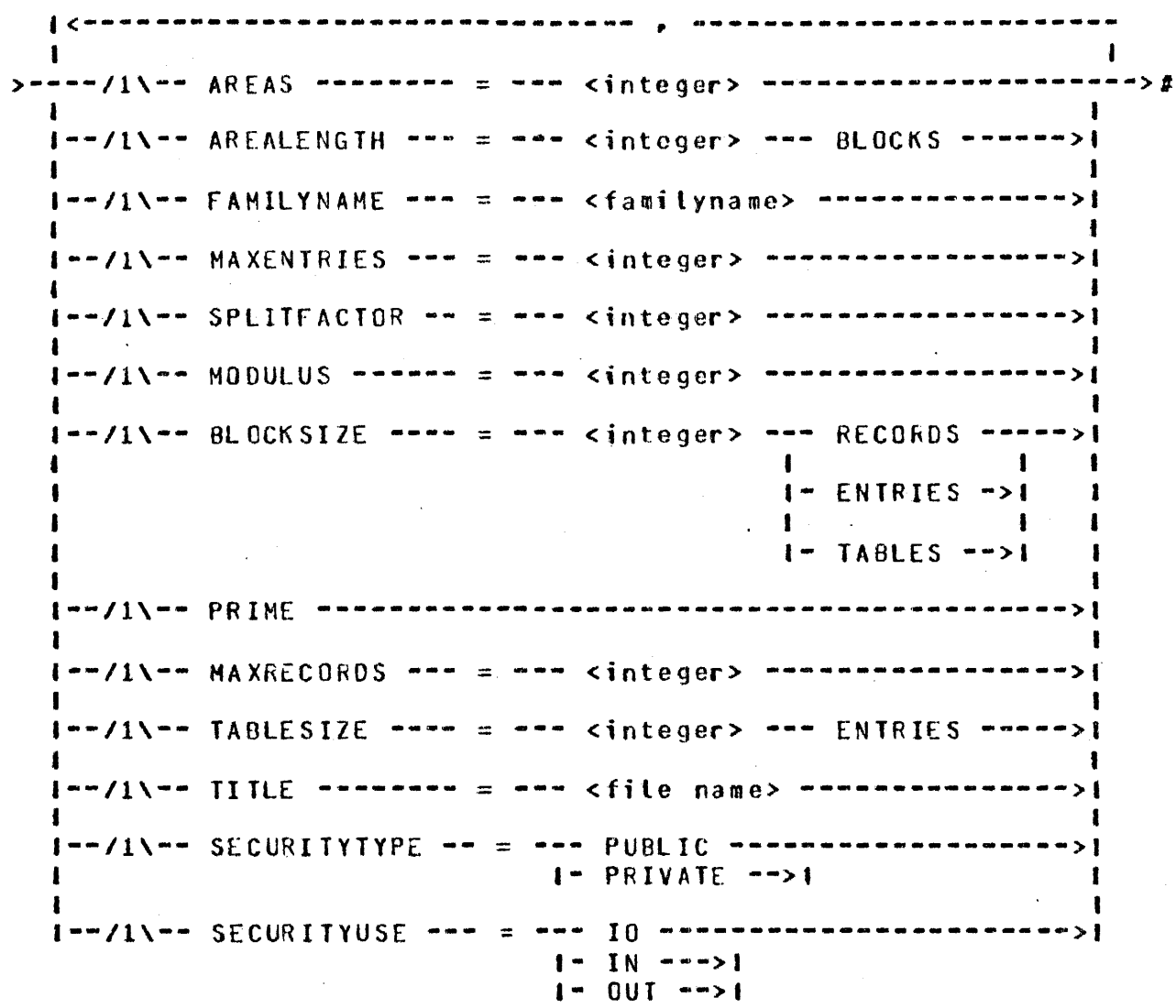
They allow the user to place all physical attributes at the end of his DASDL source input and must refer to structures that have been previously defined.

Any option appearing in a <physical attribute specification> will override the same option which was specified in the structure definition or a preceding <physical attribute specification>.

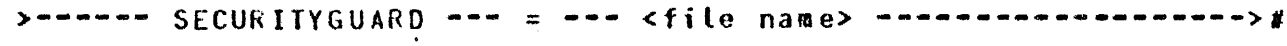
The <physical attribute specification> may be declared at the disjoint level only, but may refer to embedded structures.

The <data base name> may be that of the physical data base or one of the logical data bases.

Syntax: <physical option>



Syntax: <database physical option>



Semantics:

The <physical option> provides for specification of the physical attributes of a structure. There are four types of physical structures: standard data set, index sequential access, index random access, and list. Note that not all options are applicable to all structure types. The applicable structures are indicated in parentheses.

A. AREAS

This attribute specifies the maximum number of areas in the disk file. This attribute may be specified for all types of structures. Its value must be greater than 0 and less than or equal to 105. The default is 20 areas. (SDS, I/S, I/R, LIST)

B. AREALENGTH

This attribute specifies the maximum number of physical blocks to allocate per area on disk. (SDS, I/S, I/R, LIST)

C. BLOCKSIZE

This attribute specifies the blocking factor for the structure. The applicable units are dependent on the structure type; records for standard data sets, entries for index sequential and index random and tables for lists. (SDS, I/S, I/R, LIST)

D. FAMILYNAME

This attribute specifies the pack identifier of the file containing this structure. The reserved word PACK is synonymous with FAMILYNAME. The <familyname> may be specified as an identifier or as an alpha literal. This attribute is applicable to all structure types. The default is the object pack id from the compile statment. (SDS, I/S, I/R, LIST)

E. MAXENTRIES

This attribute specifies the maximum number of entries that may be enrolled in a set or subset. (I/S, I/R, LIST)

F. MAXRECORDS

This attribute specifies the maximum number of records that may be enrolled in a data set. (SDS, LIST)

G. MODULUS

This attribute specifies the hash-modulus to be used in providing the table number for the start of a search (i.e., the number of base tables). (I/R)

H. PRIME

When used as the attribute of a set, this attribute specifies that this set is the primary access method for the data set. The DMSII system uses this characteristic to optimize the file organization of the data base. Only one set may be specified as prime. A prime set must have the same number of areas as its data set. (I/R, I/S)

I. SPLITFACTOR

Used for index sequential and index random structures, this attribute specifies by percentage the maximum number of table entries to be split into a new table when the table size is exceeded. The minimum number is (100% - SPLITFACTOR). (I/S, I/R)

J. TABLESIZE

This attribute specifies the number of entries per table for list structures. (LIST)

K. TITLE

The TITLE attribute specifies the name of the file on disk. The default for the TITLE IS <default pack>/<data base name>/<structure name>.

L. SECURITYTYPE

See AUDIT (Section 3) for details of this attribute.

M. SECURITYUSE

See AUDIT (Section 3) for details of this attribute.

N. SECURITYGUARD

The SECURITYGUARD attribute specifies the name of a file on disk to be used to specify the security constraints on the data base. If SECURITYGUARD is not specified for a given data base, the access to that data base defaults to unrestricted. See SECURITYGUARD (Section 7) for a description of the format of a Securityguard file.

DEFAULT VALUES

Default values are computed for each applicable option if no explicit value was specified via a <physical option>. If explicit assignment is made, then the DASDL compiler will not change the assigned value.

STANDARD DATA SET

a. AREALENGTH = MAXRECORDS/BLOCKSIZE/AREAS

b. BLOCKSIZE = 1 RECORDS or the maximum number of records that will fit into 1440 bits.

c. MAXRECORDS = 10000

LIST
(Unordered List, Ordered List, Unordered Data Set, Ordered Data Set)

a. AREALENGTH = MAXENTRIES/TABLESIZE * PARENT.POPULATION/BLOCKSIZE/AREAS

b. BLOCKSIZE = 1 TABLES or the maximum number of tables that will fit into 1440 bits.

c. MAXENTRIES (or MAXRECORDS) = 10

d. TABLESIZE = 1 ENTRIES or the maximum number of entries plus the requisite control information that will fit into 1440 bits.

INDEX SEQUENTIAL ACCESS

a. AREALENGTH = 1.4 * (number of blocks necessary to represent the population)/AREAS

b. BLOCKSIZE = Square root(MAXENTRIES) ENTRIES

c. MAXENTRIES = population of the data set

d. Not PRIME

e. SPLITFACTOR = 75

INDEX RANDOM ACCESS

a. AREALENGTH = 1.3 * (MODULUS/AREAS)

b. BLOCKSIZE = 1.1 * Square root(MAXENTRIES) ENTRIES

c. MAXENTRIES = population of the data set

d. MODULUS = 1.2 * (MAXENTRIES/BLOCKSIZE) ENTRIES

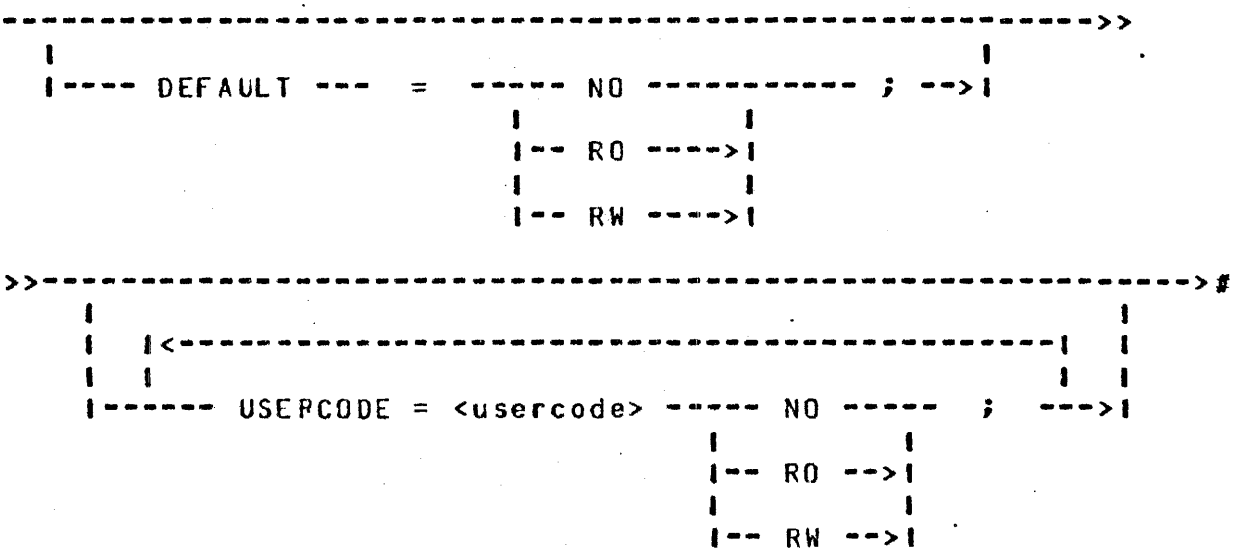
e. Not PRIME

f. SPLITFACTOR = 75

SECURITYGUARD

The Securityguard file is used to specify to the DMSII system a list of users (usercodes) who may access a given database and what type of access is permitted. This file is read by DASDL during the compilation of the data base and the access information is encoded into the DASDL generated dictionary. For this reason, changing the securityguard file after the DASDL run does not affect the data base access criteria. The access security provided by this mechanism only restricts access to the data base files via the DMSII system. Access of the data base files by any other means is handled by the regular MCPII file security rules.

Syntax: <contents of securityguard file>



Semantics:

A "%" at any point in a record denotes that the rest of that record is a comment. Only columns 1-72 of a record are scanned, and any other columns are ignored.

There are 3 access privileges. NO specifies that any attempt to open the database will be denied. RO specifies that Read Only (Inquiry) opens will be allowed. RW specifies that Read Write or Read Only (Update or Inquiry) will be allowed.

The DEFAULT statement provides the Database Administrator with a means of letting DMS know whether or not to approve database opens requested by jobs running under a usercode not listed in the Securityguard file, or by jobs not running under a usercode.

If there is no DEFAULT statement, then no access will be permitted except as specified by any subsequent USERCODE statements. This is the same as DEFAULT=NO;. The option DEFAULT=RO; will allow any job running under a usercode that was not specified in the Securityguard file (or a job not running under a usercode) to open the database Inquiry. Other open rights may then be specified in subsequent USERCODE statements.

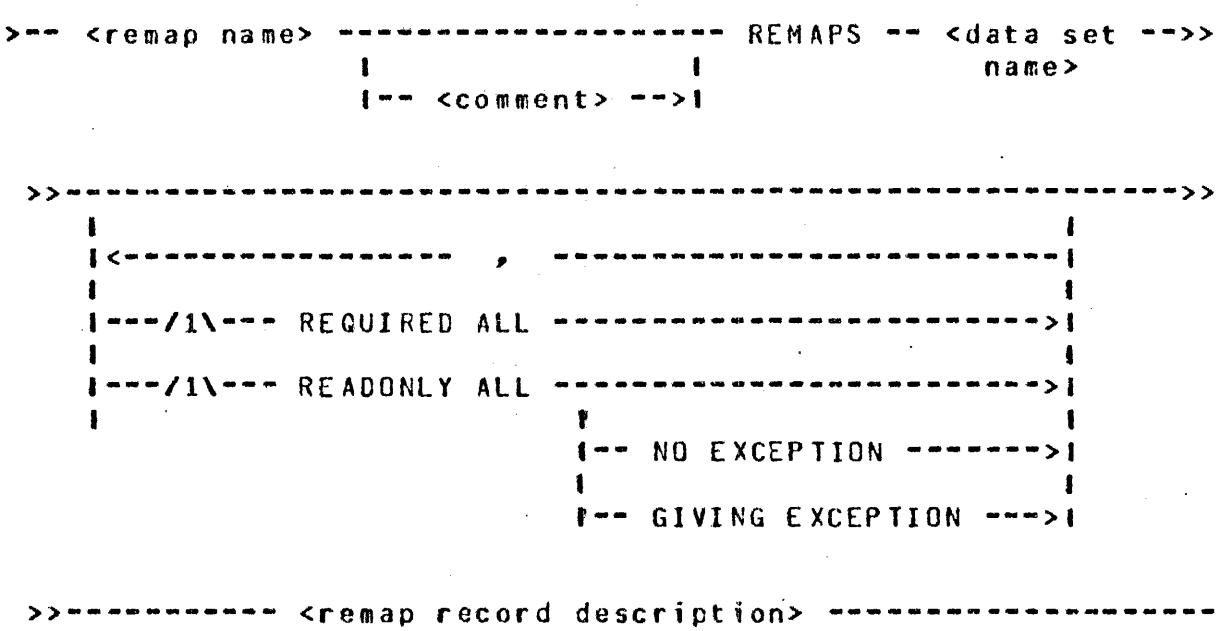
The option DEFAULT=RW; allows any job to open the database Inquiry or Update. The Database Administrator may then list the exceptions to this by adding USERCODE statements limiting access privileges for specific usercodes.

Usercode statements are the exceptions to the DEFAULT statement. A USERCODE statement is used to tell DMS that a certain usercode is to have access rights that differ from the access rights specified by the DEFAULT statement. It makes no difference if the parentheses are included with the <usercode>. If they are omitted, DASDL will supply them.

REMAPS AND LOGICAL DATABASES

REMAP STATEMENT

Syntax:



Semantics:

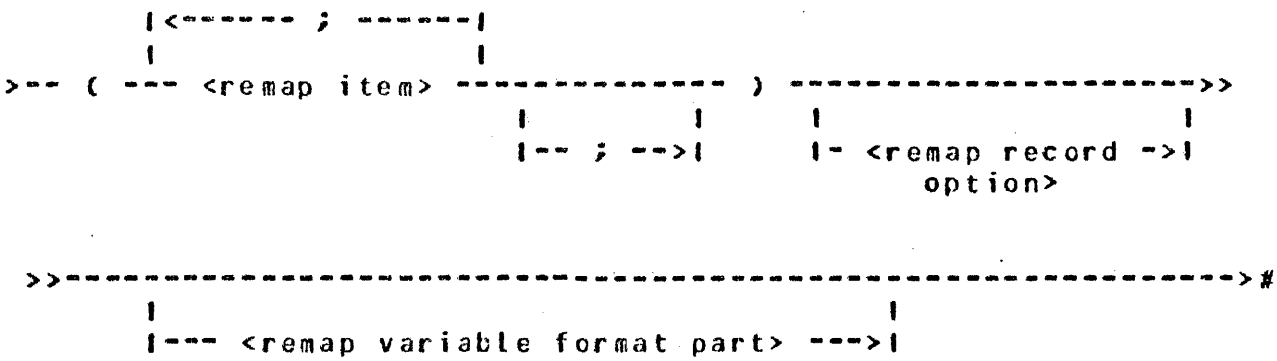
<remap name> specifies the name given to this remap of the <data set name>, which must be previously specified. Application programs may then refer to the <remap name> as if it were a data set.

REQUIRED ALL is equivalent to specifying REQUIRED for each item of the remap record.

READONLY ALL is equivalent to specifying READONLY for each item in a remap record. An exception will be returned to the host program if READONLY fields are modified, unless NO EXCEPTION has been specified.

Remap Record Description

Syntax: <remap record description>

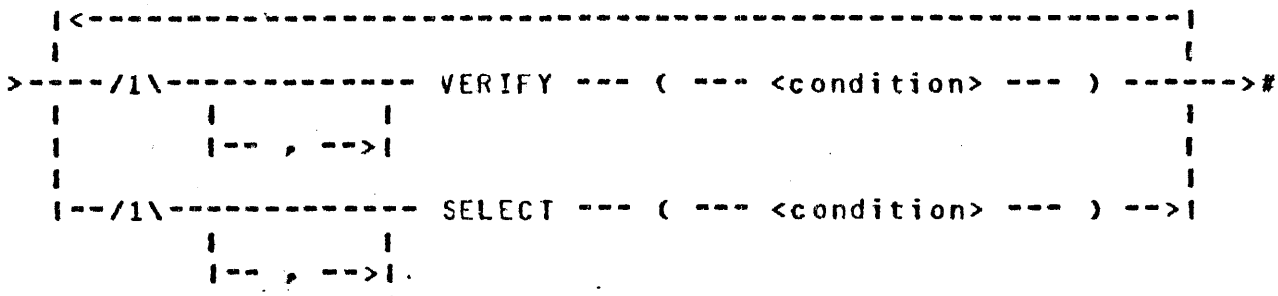


Semantics:

The <remap record description> provides a means to specify the format of records in the remapped data set. Items must have appeared in the same part (fixed part or variable part) of the original data set. Any and all items from the original data set may be omitted from the remap.

Remap Record Option

Syntax: <remap record option>



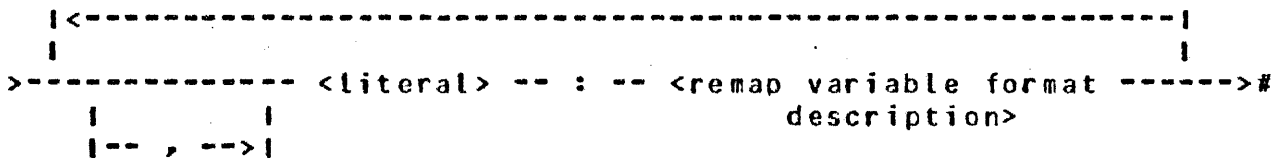
Semantics:

Only records satisfying the SELECT condition will be returned to host programs.

For STORE statements, records must meet both the VERIFY condition in the remap record description and the VERIFY condition in the original data set description in order to be stored in the data set.

Remap Variable Format Part

Syntax: <remap variable format part>

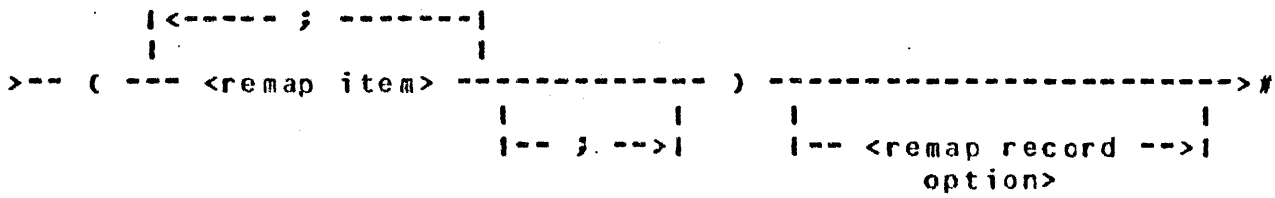


Semantics:

Allows specification of the variable format portion of the remap data set record. The <literal> must match a <literal> in variable format specification of the original data set. The RECORD TYPE item from the original data set must have been remapped in the fixed format portion of the remap record.

Remap Variable Format Description

Syntax: <remap variable format description>

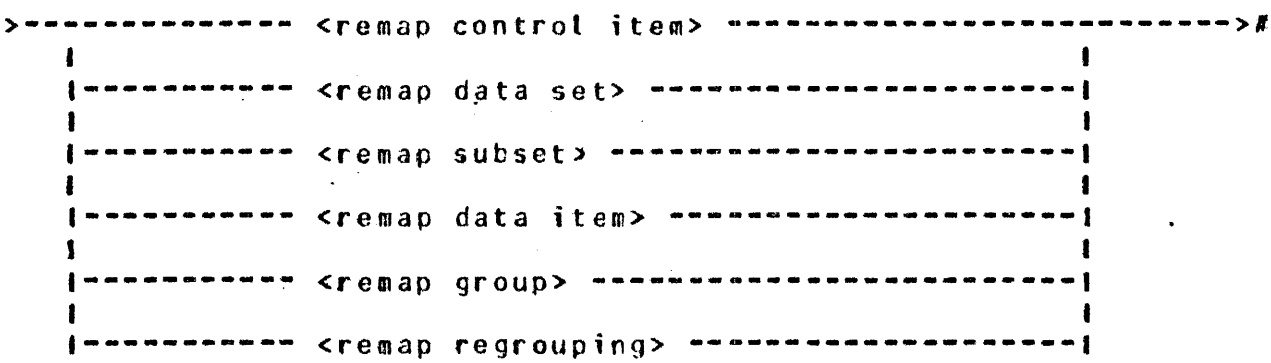


Semantics:

Only items appearing in appropriate variable formats of the original data set may be included in such a declaration.

Remap Item

Syntax: <remap item>

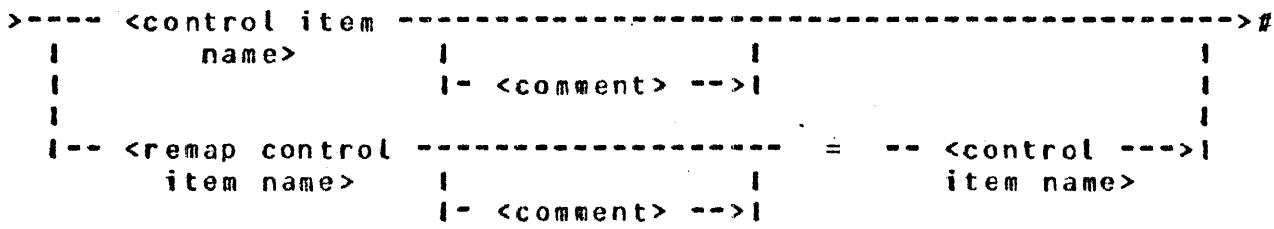


Semantics:

Items from the original data set are included in the remap data set through the use of the remap item.

Remap Control Item

Syntax: <remap control item>

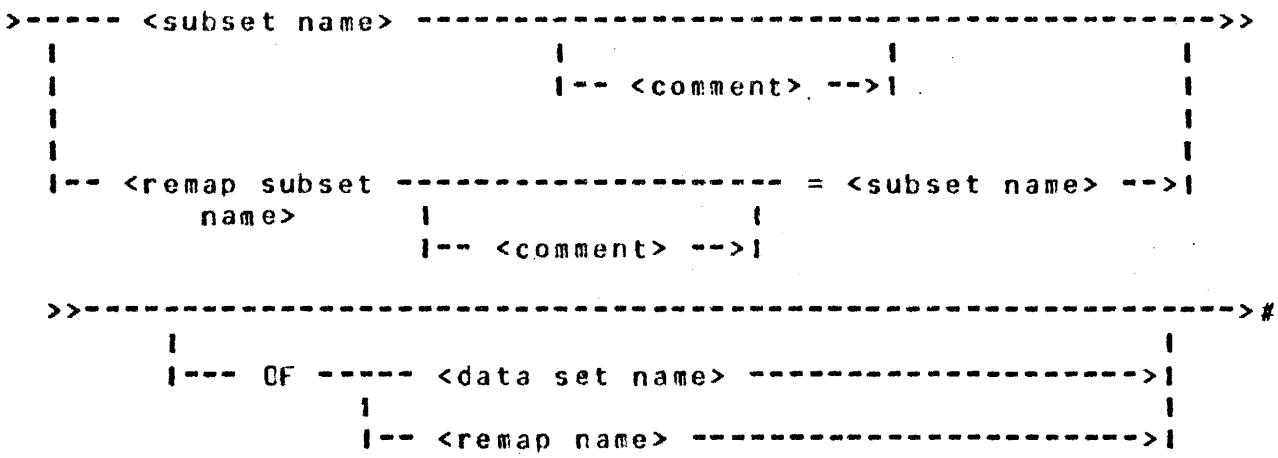


Semantics:

The <remap control item> allows control items from the original data set to be included in the remap. If a new name is desired for the <remap control item> then the <remap control item name> option may be used. The <remap control item name> must be unique in both the original data set and the remap. A control item may be remapped only once in a given remap. RECORD TYPE control items must always be remapped.

Remap Subset

Syntax: <remap subset>

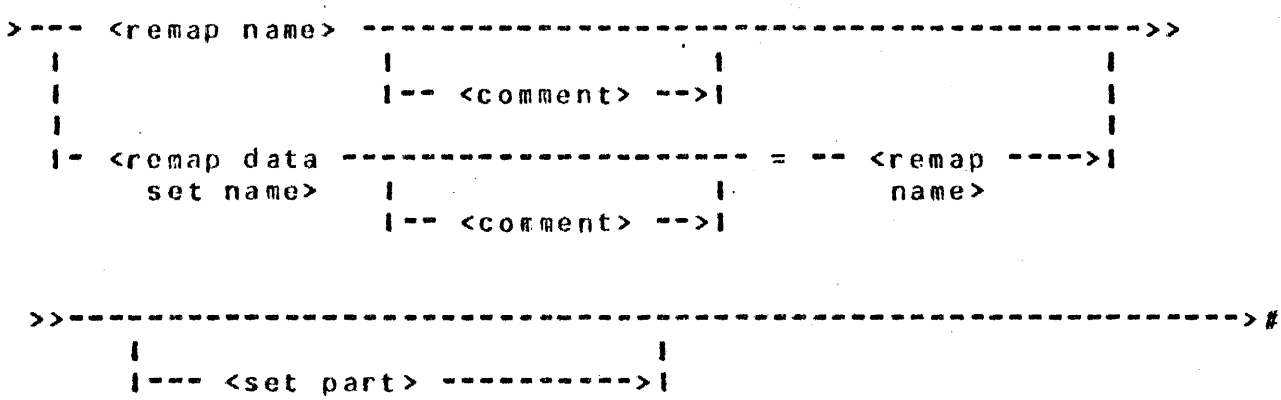


Semantics:

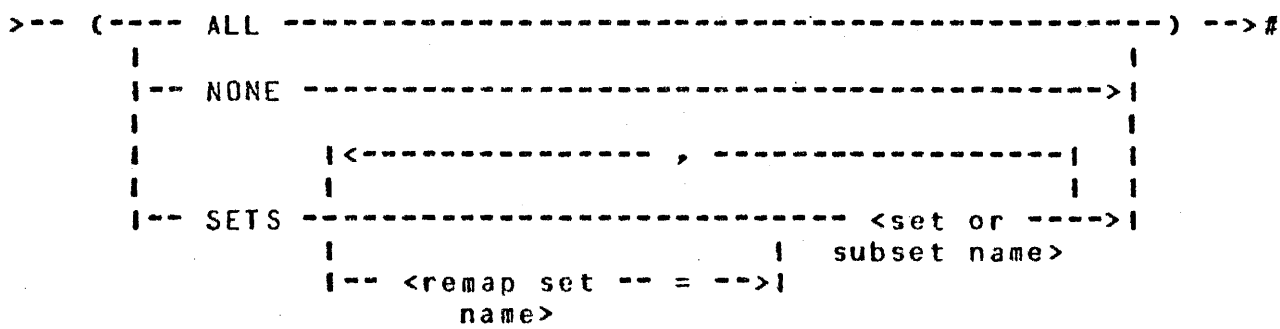
The <remap subset> allows the inclusion of manual subsets in a remap. The object data set may be specified optionally. If the object is a remap, then it must be a remap of the original object data set. If a new name is desired for the <remap subset> then the <remap subset name> optior may be used. The <remap subset name> must meet the uniqueness criteria for structures.

Remap Data Set

Syntax: <remap data set>



Syntax: <set part>



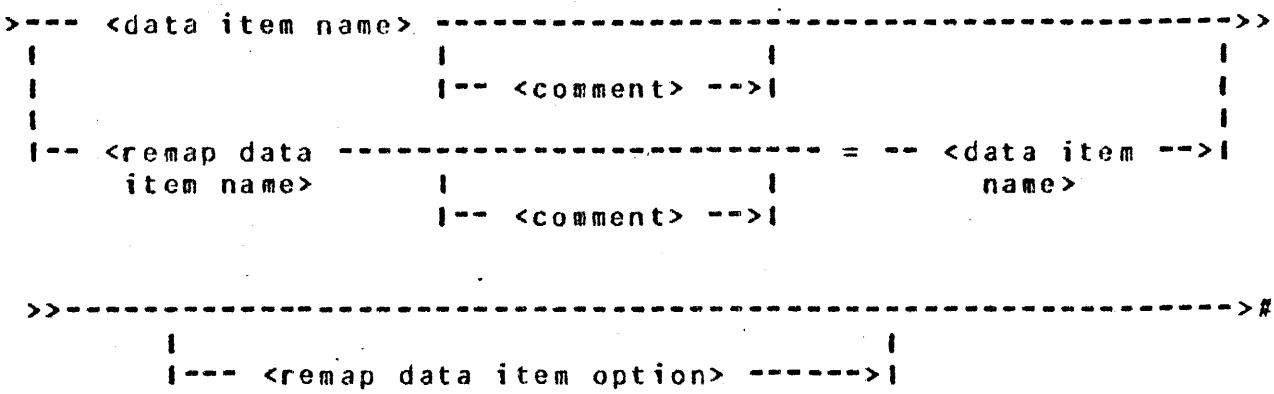
Semantics:

The <remap data set> allows the inclusion of a remap of an embedded data set in the remap. The remap of the embedded data set must have been declared in the original data set. The <remap data set name> option allows renaming of the remap. The <remap data set name> must meet the uniqueness criteria for structures. A remap may be included only once in a given remap.

The optional set part allows inclusion of all, none or only specified sets and subsets in the remap. The sets may be renamed with the <remap set name> option. The name assigned to a set must meet the uniqueness criteria for structures. If the set part is omitted, then ALL is assumed.

Remap Data Item

Syntax: <remap data item>

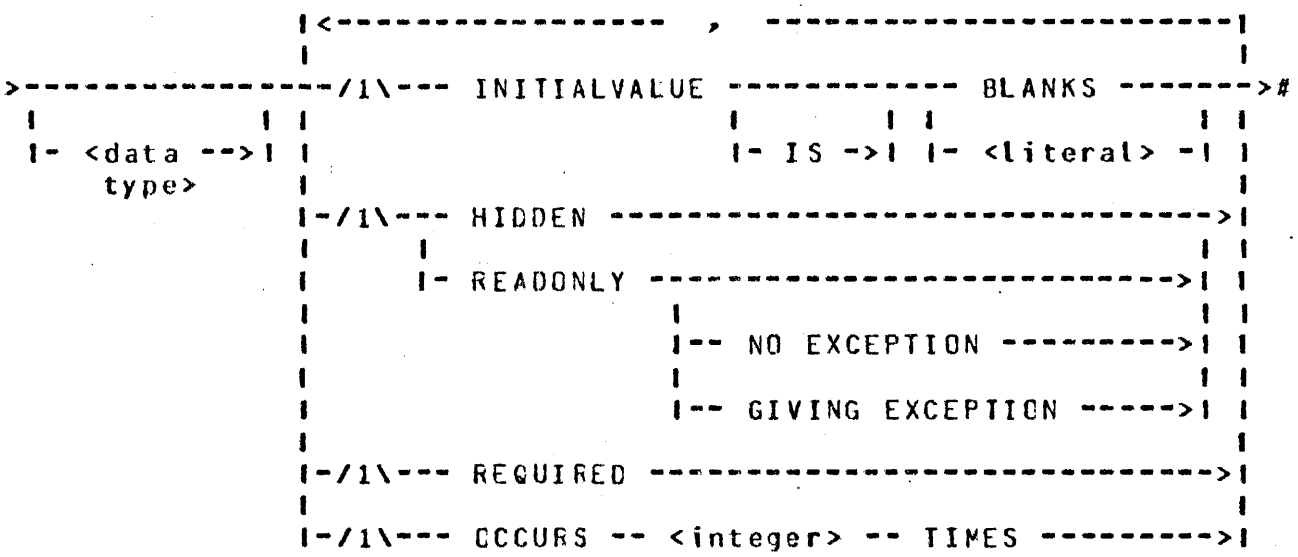


Semantics:

The <remap data item> allows data items from the original data set to be included in the remap. If a new name is desired for the <remap data item> then the <remap data item name> option may be used. The <remap data item name> must be unique in both the original data set and the remap. A data item may be remapped only once in a given remap.

Remap Data Item Option

Syntax: <remap data item option>



Semantics:

The <remap data item option> is for the purpose of respecifying the options for the data item being remapped. Any option which is not specified will default to the value of the original item's option.

If <data type> is specified, it must match exactly the original description (it is for documentation purposes only).

If the OCCURS option is specified, the value of the <integer> must be less than or equal to the original declaration.

The HIDDEN option documents the absence of a field from this remap. The HIDDEN field does not exist in the record presented to the user; however, an INITIALVALUE may be specified, and the system will initialize the field on a CREATE. HIDDEN items may also be used in SELECT and VERIFY conditions.

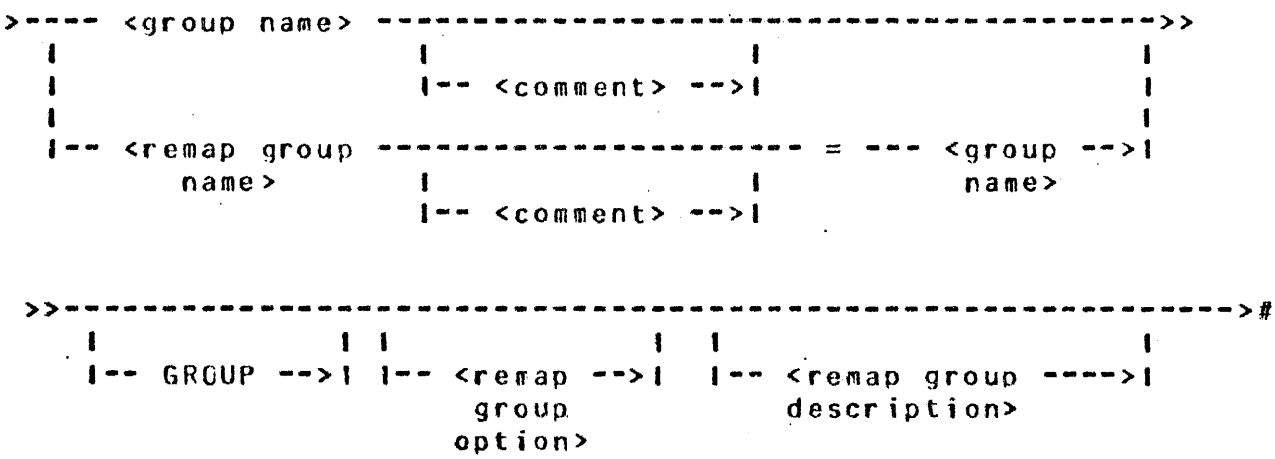
The READONLY option specifies that the item may not be altered by the host program. If such a field is altered, an exception will be returned to the host program, unless NO EXCEPTION has been specified.

The INITIALVALUE and REQUIRED options are identical to the options for data sets. <remap data item>s which are REQUIRED in the original data set are always checked for presence, regardless

of whether REQUIRED is specified or not. The INITIALVALUE specified for a <remap data item> will have precedence over an INITIALVALUE for the original <data item>.

Remap Group

Syntax: <remap group>



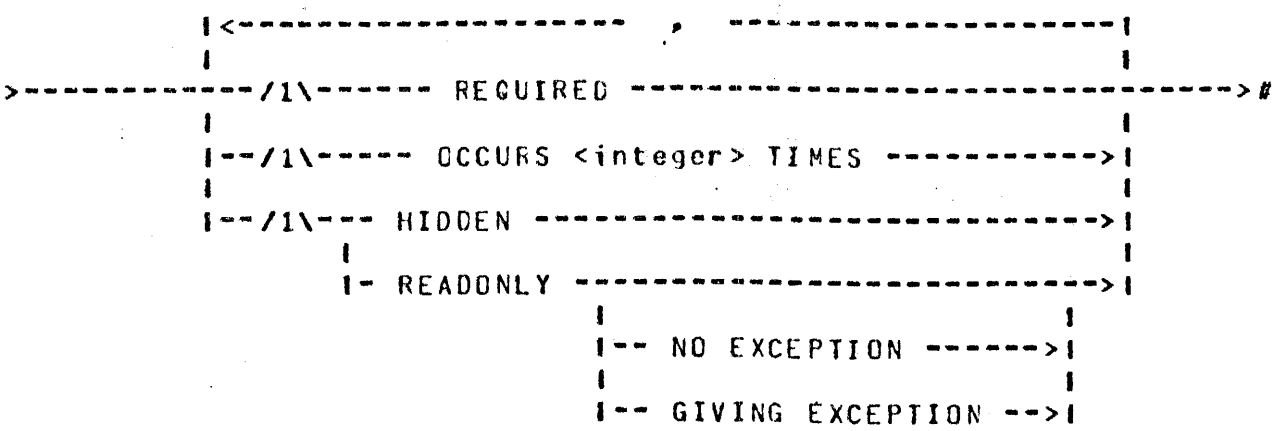
Semantics:

The <remap group> allows group items from the original data set to be included in the remap. If a new name is desired for the <remap group> then the <remap group name> option may be used. The <remap group name> must be unique in both the original data set and the remap. A group may be remapped only once in a given remap.

If the <remap group description> option is not used, then all subfields of the group are implicitly remapped.

Remap Group Option

Syntax: <remap group option>



Semantics:

The <remap group option> is for the purpose of respecifying the options for the group being remapped. Any option which is not specified will default to the value of the original group's option.

If the OCCURS option is specified, the value of the <integer> must be less than or equal to the original declaration.

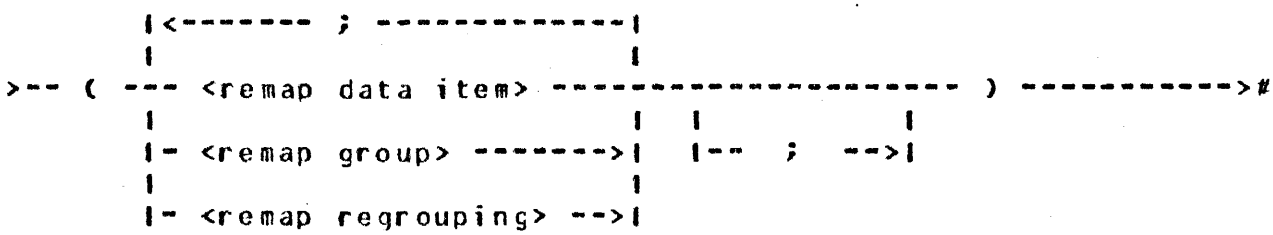
The HIDDEN option documents the absence of a group item and all its sub items from this remap. The HIDDEN group item is not in the record presented to the user. HIDDEN items may be used in SELECT and VERIFY conditions.

The READONLY option specifies that the group item and all its sub items may not be altered by the host program. If such fields are altered, an exception will be returned to the host program, unless NO EXCEPTION has been specified.

The REQUIRED option is identical to the option for group items. Items which are REQUIRED in the original data set are always checked for presence.

Remap Group Description

Syntax: <remap group description>



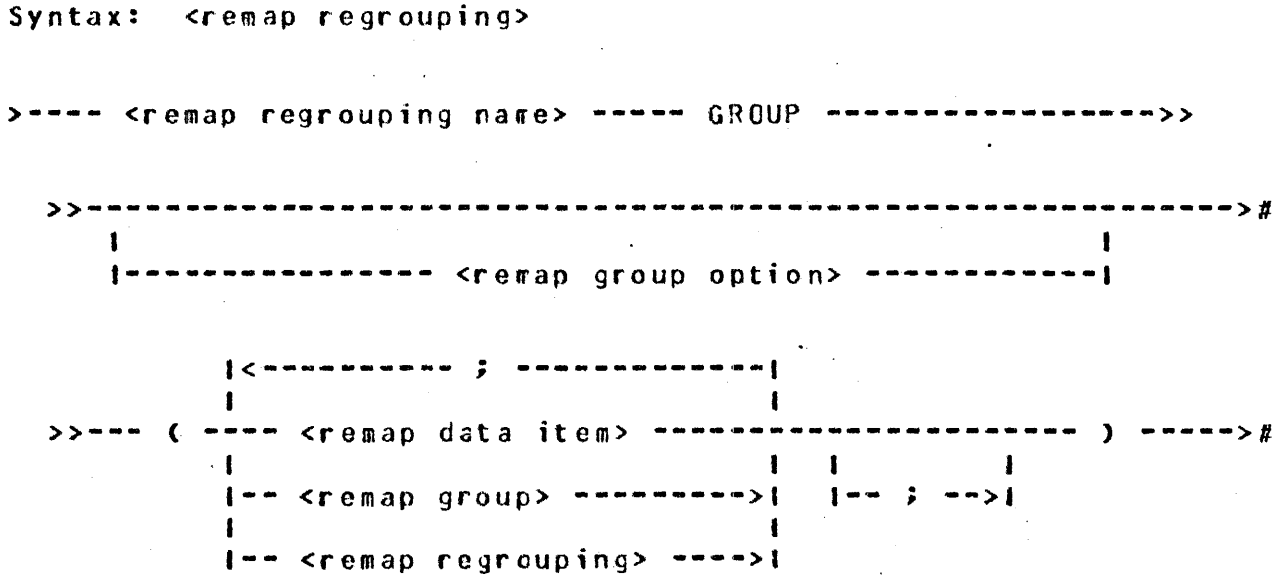
Semantics:

The <remap group description> allows for the redescription of items within a group. Only items which are declared in the original group may be remapped in the <remap group description>. All items must be respecified such that their number of subscripts does not change and the sizes of all subscripts do not exceed the original declared sizes.

If the <remap group description> is omitted, all items in the original group will be remapped without change of name or options.

Remap Regrouping

Syntax: <remap regrouping>



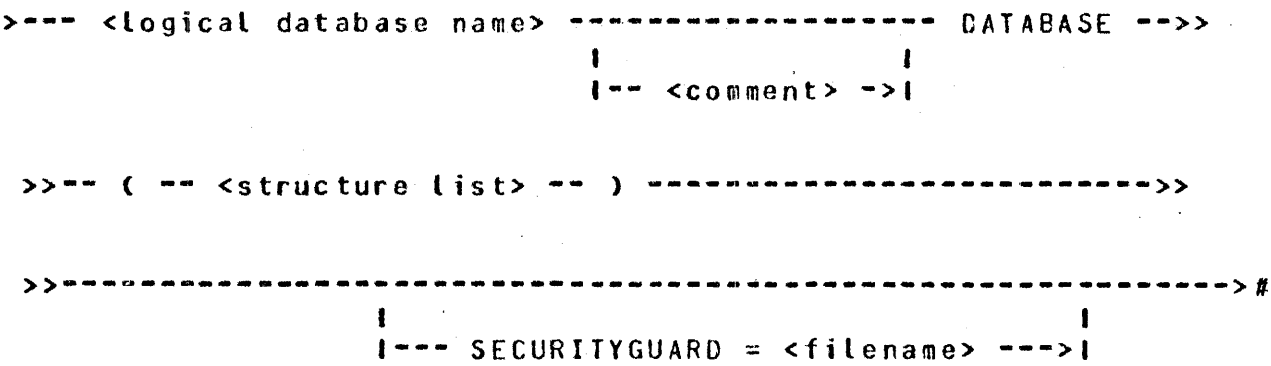
Semantics:

The <remap regrouping> allows for restructuring of the data in the remap. The <remap regrouping name> must be unique in both the original data set and the remap.

The OCCURS option must be specified in the <remap group option> such that all items remapped in the <remap regrouping> have the same number of subscripts as the items in the original data set and the sizes of all subscripts do not exceed the original declared sizes.

LOGICAL DATABASE

Syntax: <logical database>



Semantics:

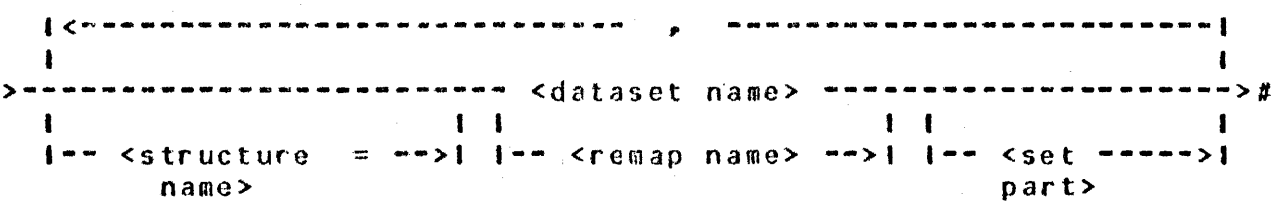
The <logical database name> specifies the name of a logical database. <logical database>s control which database structures can be accessed and how these structures can be used. When <logical database>s are declared in DASDL, the <data set>s, <remap>s, <set>s, <subset>s and <access>es which are to be included in each <logical database> are listed. Host language programs may invoke structures selectively from a <logical database>. Programs can only use the structures contained in the <logical database> they invoke, thus the database administrator may control which structures are accessed by controlling which <logical database> is used.

A SECURITYGUARD file may be assigned to each <logical database>. The Securityguard file specifies who may open the <logical database> and how they may access it. The Securityguard file is read by DASDL when the database is compiled and the access restrictions are recorded in the dictionary. See SECURITYGUARD (Section 7) for a description of the format of a Securityguard

file.

Structure List

Syntax: <structure list>



Semantics:

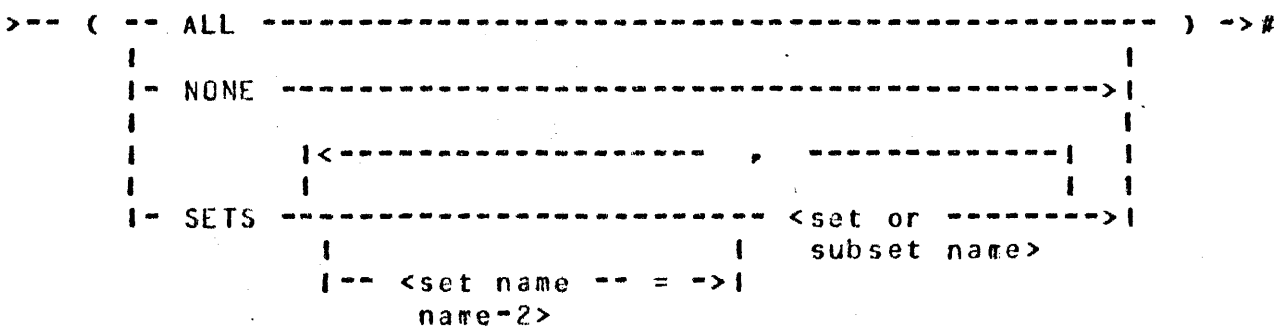
The <structure list> specifies which <data set>s, <remap>s, <set>s, <subset>s and <access>es are to be included in the <logical database>.

The <structure name> option changes the name of a <data set> or <remap>. When the <logical database> is invoked and listed in a user program, the new name replaces the original name. All user programs which invoke the <logical database> refer to the structure using the new name.

All <data set>s and <remap>s named in the <structure list> must be disjoint. If a <data set name> is specified, all of its embedded <data set>s are automatically included in the <logical database>, and embedded <remap>s are excluded. When a <remap name> is specified, the embedded structures named in the <remap> are included in the <logical database>.

Set Part

Syntax: <set part>



Semantics:

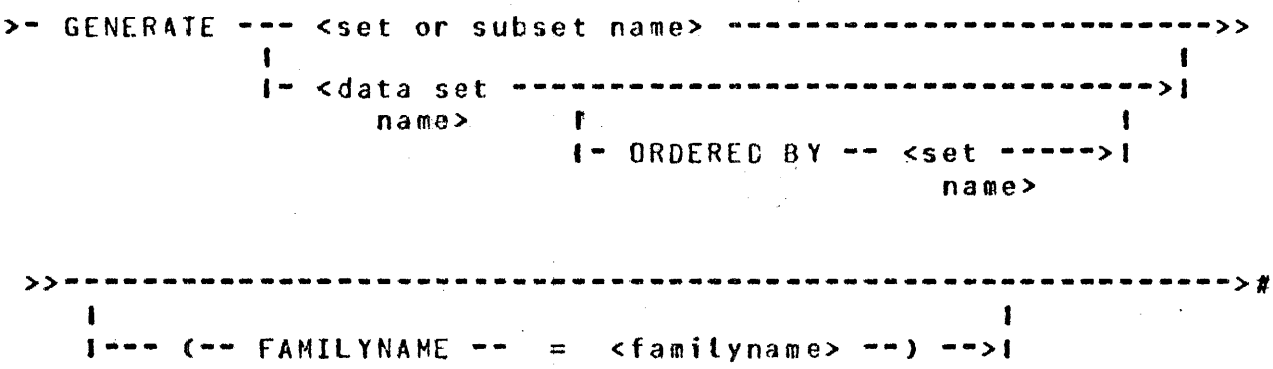
The <set part> specifies which disjoint <set>s, <subset>s and <access>es are included in the <logical database>. If the <set part> is not present, or if ALL is stipulated, all disjoint <set>s, <subset>s, and <access>es are included. If <set>s, <subset>s or <access>es are specified, they must refer to <data set>s or <remap>s which are included in the <logical database>. The <set.name.2> option can be used to change the name of a <set>, <subset> or <access> in the <logical database>.

REORGANIZATION STATEMENTS

The reorganization statements are part of the general reorganization capability of the DMSII system. For more details see the B1800/B1700 DMSII Reorganization document, P.S. 2219 0540.

GENERATE STATEMENT

Syntax:

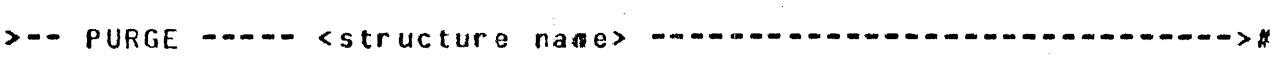


Semantics:

The GENERATE statement causes the system to garbage collect the structure specified. The GENERATE statement may only be used if the $REORGANIZE option is set.

PURGE STATEMENT

Syntax:

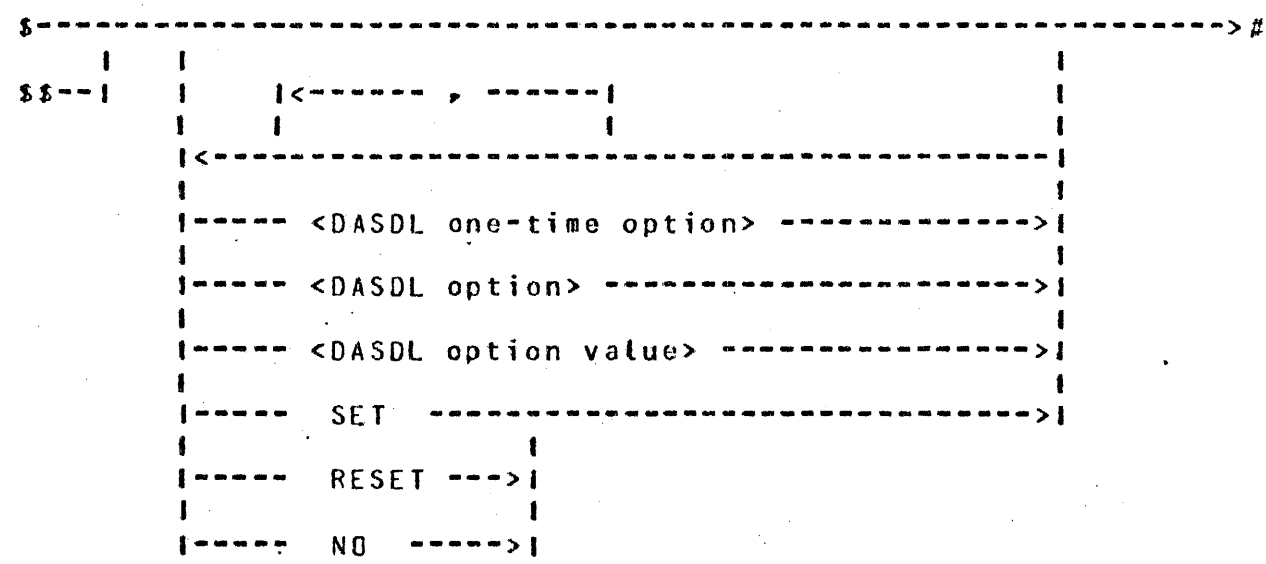


Semantics:

The PURGE statement causes the system to remove all elements from a structure. The PURGE statement may only be used if the $REORGANIZE option is set.

COMPILER $ OPTIONS

The following describes the general syntax of the DASDL Compiler Control Options.



GENERAL SEMANTICS

1. Every DASDL compiler control option card image must have a dollar sign ($) character in column 1. Columns 73-80 may be used as a sequence field. The card image is interpreted from left to right beginning with the first character following the dollar sign character(s).

2. Any DASDL option card image beginning with two consecutive adjacent dollar sign characters ($$) denotes a permanent DASDL option card image, which will be included in any new source file generated via the "NEW" option. A single dollar sign ($) character denotes a temporary option, the life of which is limited to that of the current compile.

3. DASDL one-time opions may be SET or RESET only once, and such setting or resetting must occur before any DASDL source images are processed. All other DASDL compiler control options may be SET or RESET any number of times and anywhere within the DASDL source file.

4. If a DASDL compiler control option appears without an explicit "SET", "RESET", or "NO", it is implicitly enabled.

5. Parameters may or may not be required for any particular option. See the explanation for the item involved to determine the suitability of parameters.

   NOTE: Any option requiring parameters must be the last option to appear on a DASDL compiler control card image.

DASDL ONE-TIME OPTIONS

Parameters are not allowed unless otherwise specified. Default setting is disabled for all options unless noted otherwise.

CONVERT — This option specifies that the internal format of an existing data base dictionary must be modified to reflect the format required by the current level of the MCP. All dictionaries created previous to 8.0 must be converted to the 8.0 dictionary format in order to run on an 8.0 MCP. As conversion is a self-contained process, no other input is accepted. Any DASDL source images following this card will be ignored.

FILE — See STRUCTURE.

INITIALIZE — When preceeded by "NO" or "RESET", this option inhibits file initialization following DASDL source encoding. This option is intended for development use only.

MERGE — This option, when enabled, causes the DASDL compiler to merge source images from the primary input file ("CARDS") together with source images from the secondary input file ("SOURCE").

NEW — When "set", this option enables creation of a new source file ("NEWSOURCE") containing all source images and permanent DASDL compiler control options processed from the primary and secondary input files.

REORGANIZE — This option enables reorganization of an existing data base. For more details, see the B1800/B1700 DMSII REORGANIZATION document, P.S. 2219 0540.

SOURCE — This option enables the printing of the COBOL library files generated by DASDL in the output listing for this compile.

SOURCEONLY — When enabled, this option causes DASDL to regenerate COBOL and RPG library files for an existing data base without recreating the data base. No other functions are performed when this option is specified, and no other source images will be processed.

STRUCTURE — This option enables the printing of statistics from data base structure records that show the layout of records, index tables, and list structures included in this data base. Since there is a one to one correspondence between structures and files, information on files is included with the structure information.

TABLESIZE <max tablesize>

When enabled, this option causes a limit to be put on the maximum size of all indexed sequential and indexed random sets in this data base. This limit may be overridden by an explicit MAXENTRIES declaration for any individual set, but serves as a bound for the default calculation for any set not explicitly specified. The <max tablesize> is a required parameter, and indicates the maximum size in disk segments that the tables are allowed to attain. The maximum value for <max tablesize> is 45, the default value is 20.

TAPE — Used in conjunction with "REORGANIZE" to specify tape as the communication media for the data base reorganization programs. Default media for these programs is a queue file.

UPDATE <data base name>

This option specifies that the data base source description be compared to and supercede the existing data base description. UPDATE may be used in conjunction with the REORGANIZE option or by itself. When used alone, the new description of the data base may specify new structures or exclude existing structures as long as these changes do not cause any restructuring of the continuing data base. The <data base name> is an optional parameter, and specifies the name of an existing data base which is to be updated. If omitted, the data base name associated with this DASDL compile is assumed. Use of the data base name parameter has no effect on the names of the data base files being created. This option, if specified, must be the last option to appear on a compiler control card image.

VERSIONCHECK This option causes the run-time verification that the version (date and time of compile) associated with the COBOL library files of any application program attempting to access this data base is equal to that associated with the structures in the data base itself. The default setting for this option is enabled.

DASDL OPTIONS

Unless otherwise specified, all options default to disabled and do not allow parameters.

COBOL This option causes the DASDL compiler to check for COBOL reserved words which may occur as identifiers in the DASDL source. This prevents syntax errors from occuring in the COBOL library files generated by this compile.

COBOLIB <logical data base name>

This option causes DASDL to generate COBOL library files on disk for the logical data base specified. If <logical data base name> is omitted, then the name of the physical data base is assumed. The COBOLIB option is set by default for the physical

data base, but reset for all logical data bases.

CODE — This option causes the DASDL compiler to print information about the DASDL generated code. Two tables are listed.

The first table lists each code segment and its size.

The second table lists each structure by name, and for each of the up to 6 kinds of code that could be generated for a structure, the segment number of the segment that contains that particular type of code.

This is intended to be used in conjunction with SYSTEM/MARKSEGS.

DEBUG — This option causes DASDL to include reorganization control information in the output listing generated by this compile. This option is intended for development use only.

DELETE — When enabled, this option causes the DASDL compiler to discard source images from the secondary input file ("SOURCE") until disabled. It must appear on a source image in the primary input file ("CARDS"), and has no effect if "MERGE" has not been enabled. The normal merging process will not be altered, but all source images (including DASDL compiler control options) will be discarded as they are read from the secondary file. These source images will not be listed or carried forward into a new source file.

DOUBLE — When enabled, causes the DASDL output listing to be double spaced.

INCLNEW — This option, when enabled, causes any source images processed as a result of the "INCLUDE" option to be included in the new source file ("NEWSOURCE") should the "NEW" option be enabled. If "NEW" is not enabled, this option has no effect.

INCLUDE <file-id>

When enabled, this option causes DASDL to suspend the normal sequence of processing source images and accept input from the file specified by <file-id> until such input is exhausted. Normal compilation is then resumed. The <file-id> is a required parameter, and is in the normal file identifier format of <pack>/<mfid>/<fid>. This option, if specified, must be the last option to appear on the compiler control card image, and may not appear on a source image within a file processed via the "INCLUDE" option.

LIST — This option causes a listing of all source images and permanent DASDL compiler option card images to be included in the output listing for this compile. The default setting for this option is enabled.

LIST$ — This option causes a listing of all temporary DASDL compiler control card images to be included in the output listing for this compile. If the LIST option is disabled, LIST$ has no effect.

LISTINCL — When enabled, this option causes the inclusion of all source images processed via the "INCLUDE" option in the output listing for this compile. If the LIST option is disabled, LISTINCL has no effect.

PAGE — When encountered, this option immediately causes a page advance to be placed in the output listing for this compile.

RPG — This option causes the DASDL compiler to check the database description for RPG compatibility, respecting identifier size and subscripting.

RPGLIB <logical data base name>

This option causes DASDL to generate the RPG library files on disk for the logical data base specified. If <logical data base name> is omitted, then the name of the physical data base is assumed. The RPGLIB option is reset by default for the physical data base and all logical data bases.

SEQ or SEQUENCE

This option, when enabled, causes the DASDL compiler to assign new sequence numbers to the source language card images accepted for input. This option assigns the Current Sequence Base (see DASDL option values) to the current source image, and increments the Current Sequence Base by the Current Sequence Increment (see DASDL option values).

SINGLE — When enabled, this option causes the compiler output listing to be single spaced. This option defaults to enabled.

STANDARD — This option causes the DASDL compiler to enforce syntax rules as described by the Burroughs DASDL Standard.

SUPPRESS — This option causes the DASDL compiler to suppress the inclusion of warning messages in the output listing for this compile. This option performs the same function as "WARNSUPR".

VOID <seq-no>

This option, when enabled, causes all input images in the secondary source file ("SOURCE") to be discarded until <seq-no> has been exceeded by a sequence number in the secondary source file. The <seq-no> is an optional parameter and must be 8 digits in length. If omitted, only the present card image will be discarded.

WARNSUPR — When enabled, this option causes the DASDL compiler to suppress inclusion of warning messages in the output listing for this compile. See "SUPPRESS".

DASDL OPTION VALUES

These values are used by the DASDL compiler to control resequencing, and may appear anywhere on a compiler control card image. They have no effect if the "SEQ" option is not enabled.

Current Sequence Base

This is a 1 to 8 character string of digits signifying the current base sequence number from which resequencing will take place. The default value for this entity is "00001000".

Current Sequence Increment

This is a character string comprised of a "+" character followed by 1 to 8 digits signifying the current increment to be used in resequencing. The default setting for this entity is "+1000". Spaces may optionally appear between the "+" character and the sequence increment.

DASDL FILES

CARDS	This is the primary source file for the DASDL compiler. It is the first file to be read, and may contain the entire set of source images to be processed. Optionally, (if "MERGE" is enabled) it may contain patches to be applied to the secondary input file ("SOURCE").
COBOL.LIB	The COBOL library files created by DASDL are written serially to this file.
DMS.DICT	This file is the data base dictionary associated with the data base to be created by this DASDL compile.
DMS.OLD.DICT	This file is an existing data base dictionary which will be used by the DASDL compiler to create a new data base dictionary.
ERROR.LINE	This is the error message output file used in conjunction with compiles via CANDE. When a compile is initiated from a terminal through CANDE, or DASDL is executed with SW1 = 1, all error messages will be dumped to this file.
GUARDFILE	This is a serial input file containing security access restrictions for the data base.
LIBRARY	This file contains source statement images to be processed via the "INCLUDE" option.
LINE	This is the output listing file, containing a list of processed source images. Its contents are controlled via the "LIST", "LIST$", and "LISTINCL" options.
NEWSOURCE	This is the output symbolic file, a file containing those source images selected by DASDL in accordance with the "NEW" and "INCLNEW" options.
REORG.CODE	This is an output file used for the creation of the REORG.READER and REORG.WRITER code files.
REORG.FILE	This is an input file used for the creation of the REORG.READER and REORG.WRITER code files.
RPG.LIB	The RPG library files created by DASDL are written

serially to this file.

SCL.FILE — This is a random output file used for the creation of the data files for the data base being processed.

SOURCE — This file is the secondary input file, which contains previously stored DASDL source images. It is not used unless the "MERGE" option is enabled.