**Burroughs Corporation**

COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

2219 0540

B1800/B1700 DMSII REORGANIZATION

# PRODUCT SPECIFICATION

| REV LTR | REVISION ISSUE DATE | APPROVED BY | REVISIONS |
|---|---|---|---|
| A | 1/16/78 | | Initial Release - Mark 7.0 Level |
| B | 2/12/79 | | Major Revision - Mark 8.0 Level |

26

## TABLE OF CONTENTS

## INTRODUCTION

The purpose of reorganization is to allow the user to alter the physical and/or logical description of an existing data base, with maximum system assistance in the restructuring of the data base yet with a minimum impact on user programs which reference the data base.

The UPDATE capability first provided in the 6.0 release, allows the user to add or delete structures from the database as long as it requires no restructuring of the continuing database. The only affected programs are those programs which use deleted information or desire to use the new information. These programs have to be changed and recompiled.

To use the UPDATE capabilities, the user compiles a description of the new data base. This description is preceded by an $UPDATE card, to indicate to the Data and Structure Definition Language (DASDL) that this is an existing database and a comparison of the old and new descriptions is required. As long as the differences between the two descriptions satisfy the rules of legal changes, then the new dictionary and library files supersede the old ones. No changes are made to the database. The version stamps of all existing structures are unchanged so that user programs are unaffected unless they access deleted structures, in which case they get a VERSION ERROR when they attempt to open the database.

The reorganization capability allows the user to redescribe the database (with any implicit physical or logical changes) and/or to explicitly request that special reorganization functions be performed on the database.

Reorganization is accomplished by first running a DASDL compilation with a $REORGANIZE card and the redescription of the database and/or requests for special reorganization functions. DASDL produces a special reorganization control file which describes the particular changes that must be made to the database to accomplish the reorganization. DASDL then makes copies of the reorganization programs, binding them to this particular database and appending a DMS path dictionary. The user executes these special copies of the reorganization programs, which then accomplish the reorganization process.

## READER AND WRITER PROGRAMS

There are two reorganization programs, the reader (DMS/REORG.READ) and the writer (DMS/REORG.WRIT). Both are written in SDL. When DASDL copies and binds these programs to the particular database, it also changes the names to <database name>/REORG.READ for the reader and to <database name>/REORG.WRIT for the writer. Reorganization is actually initiated whenever the user desires, by executing the reader. The reader zip-executes the writer, extracts the information in the old database, using the MCP access routines in most instances (via the COMMUNICATE operator), and then places this information in a queue file. The WRITER program accepts the information from the queue file and places it into the new database using, in most instances, the MCP access routines. At the completion of the reorganization, the new database dictionary is marked to indicate that the new database is valid and the old database dictionary is removed.

If a lack of memory or disk space precludes these two programs and the two databases from being active at the same time, then tape may be used as the intermediate medium instead of the queue file. This is requested by including a $TAPE card in the DASDL compilation. The two programs will then execute serially, instead of the normal simultaneous execution. Disk may also be used, by OU-ing the tape file to disk at run time.

If any DMS exception occurs during the execution of either of these two programs (e.g., if duplicates are not allowed on a new set and the data in the old database includes duplicates), then the reorganization will terminate, the new database will be marked unusable and the old will be marked as valid (but not reorganized).

If reorganization accesses corrupted data from the old database or if reorganization aborts (program failure or system failure), then the old database may become unusable (see ABNORMAL CONDITIONS section). For this reason, it is advisable to dump (copy) the entire database before running reorganization.

AUDIT and RECOVERY will be unable to operate through an UPDATE or a reorganization. For this reason, and for backup, the entire database should be saved after a reorganization, also.

## RELATED DOCUMENTATION

| NAME | NUMBER |
|---|---|
| DASDL | P.S. 2219 0433 |
| DMS AUDIT and RECOVERY | P.S. 2219 0532 |
| B1800/B1700 MCPII | P.S. 2212 5462 |
| B1800/B1700 Software Operational Guide | 1068731 |

# DATA BASE REDESCRIPTION RULES

The following rules identify those changes that may be made when redescribing a database for reorganization. Rules marked with an asterisk (*) require a version stamp change and the necessary recompilations (see VERSION CHECKING section). Further information on rules 1 through 8 is in the DATA TRANSFORMATIONS section.

*1. Items may be added to existing data sets (fixed or variable format parts). These new items may be used as key items or may be required only if an initial value was specified.

*2. Items may be deleted from existing data sets (fixed or variable format parts).

*3. Items sizes may be increased or decreased. This includes both the fraction and integer part of numbers. (Key items of ordered manual subsets may not be changed.)

*4. Signs may be added or dropped on numbers. (Key items of ordered manual subsets may not be changed.)

*5. Occurences may be increased or decreased.

*6. Item types may be changed. (Key items of ordered manual subsets may not be changed.)

*7. Groupings and/or levels may be changed. The items must be used within the scope of the same data sets in the old and the new database.

*8. Items may move from the fixed format part to a variable format part and visa-versa.

*9. The ordering of the items may be changed.

10. Sets and automatic subsets may be added.

*11. Sets and automatic subsets may be deleted.

12. Sets and automatic subsets may change their duplicates clause.

*13. Sets and automatic subsets key items may be changed.

*14. Sets and automatic subsets ascending and descending may be changed.

*15. Index sequential sets may be changed to index random sets or automatic subsets.

*16. Index random sets may be changed to index sequential sets or automatic subsets.

*17. Automatic subsets may be changed to sets.

18. Embedded data sets and manual subsets may be added.

*19. Embedded data sets and manual subsets may be deleted.

*20. Ordered data sets may be changed to unordered.

*21. Unordered data sets may be changed to ordered.

*22. Manual subsets with keys may be changed to manual subsets without keys.

23. All of the physical attributes may be changed. These include:

PRIME
MAXENTRIES
MAXRECORDS
TABLESIZE
MODULUS
BLOCKSIZE
AREALENGTH
SPLITFACTOR (reorganization not required)
AREAS
FAMILYNAME
TITLE
SECURITYTYPE
SECURITYUSE

A TITLE in the old database may only be used in the new database for the same file or for a file that is being reorganized.

*24. The conditions for automatic subsets may change.

25. The conditions for VERIFYs may change.

Note: The comparison on conditions is for an equivalent expression, not for an identical expression.

## DATA TRANSFORMATIONS

During reorganization, data items within a data set may change in size, type, offset and number of occurrences, subject to certain restrictions which are discussed below.

Data items may be added or deleted from the description of a data set. When a data item is deleted, the data associated with that item is removed from all records in the data set. When a data item is added, a data field containing the initial value of the data item, if specified or null, is inserted into all records in the data set. Items which are added and do not have an initial value specified may not be used as keys or be required fields.

Data item sizes may be changed. If the old size is greater than the new size, then data will be truncated from the item. This condition is detected by DASDL and a warning message is given.

Data items may add or delete the sign field. Deletion of the sign field is detected by DASDL and a warning message is given.

OCCURS nesting may go to three levels in DASDL. A data item's number of levels of nesting may change. A data item's number of occurrences may change. If the number of occurrences changes without changing the number of levels, then each level is preserved and each subscript remains as it was. If the number of occurrences decreases in the new database, only the first n occurrences will be moved to the new record, where n is the number of occurrences of the data item in the new data set record. This condition is detected by DASDL and a warning message is given. If the number of occurrences increases in the new database, only the first m occurrences will have data moved into them from the old data set record, where m is the number of occurrences of the data item in the old data set record.

In transforming PERSON from database A to database B in Figure 3-1, the transformation would be as shown in Table 3-1a, and there would be three slots in database B without valid data. Going from B to A, three occurrences of PERSON would be lost and a warning message would be given.

If the number of levels changes, as from database A to database C in Figure 3-1, then the structure is recreated linearly and the old subscripts may have no relation to the new, as in the mapping shown in Table 3-1b.

```
JOB GROUP OCCURS 3 TIMES
    ( ...
      PERSON ALPHA (10) OCCURS 2 TIMES;
      ... );
```

Figure 3-1a: Example database A

```
JOB GROUP OCCURS 3 TIMES
    ( ...
      PERSON ALPHA (10) OCCURS 3 TIMES;
      ... );
```

Figure 3-1b: Example database B

```
JOB GROUP OCCURS 2 TIMES
    ( ...
      JOB ORDER OCCURS 2 TIMES
          ( ...
            PERSON ALPHA (10) OCCURS 2 TIMES;
            ...)
      ... );
```

Figure 3-1c: Example database C

| A | B |
|---|---|
| (1,1) | (1,1) |
| (1,2) | (1,2) |
| --- | (1,3) |
| (2,1) | (2,1) |
| (2,2) | (2,2) |
| --- | (2,3) |
| (3,1) | (3,1) |
| (3,2) | (3,2) |
| --- | (3,3) |

| A | C |
|---|---|
| (1,1) | (1,1,1) |
| (1,2) | (1,1,2) |
| (2,1) | (1,2,1) |
| (2,2) | (1,2,2) |
| (3,1) | (2,1,1) |
| (3,2) | (2,1,2) |
| --- | (2,2,1) |
| --- | (2,2,2) |

Table 3-1a: Mappings of PERSON between Figure 3-1b databases

The grouping and/or levels of data items may be changed, subject to the following restrictions:

1. Regrouping of items may not cause data to be duplicated. For example:

```
        old                           new
        ---                           ---
        A GROUP                       A ALPHA(2);
        (B  ALPHA(1);                 B ALPHA(1);
         C  ALPHA(1));                C ALPHA(1);
```

   is forbidden. In this example, the oata represented by A are duplicatec in the new definition, since B and C both contain data contained in the new A.

2. Regrouping of items may not cause multiple mapping of information into an item. This would occur if the new definition were transformed into the old definition in the example above.

Item types may be changed. The only restriction imposed here is that a decimal or signed decimal item may not be changed to be an elementary alpha item (this is a COBOL rule).

When DASDL detects that an item must be transformed, it effectively generates a MOVE which conforms to the COBOL conventions.

| | | truncation or .. | | | | | |
|---|---|---|---|---|---|---|---|
| MOVE | | space fill on right | zero fill on right | zero fill on left | truncate sign | generate positive sign | translate |
| FROM | TO | | | | | | |
| group | group | x | | | | | |
| group | alpha | x | | | | | |
| group | integer | | x | | | | x |
| group | signed int. | | x | | | x | x |
| group | decimal | | x | | | | x |
| group | signed dec. | | x | | | x | |
| alpha | group | x | | | | | |
| alpha | alpha | x | | | | | |
| alpha | integer | | | x | | | x |
| alpha | signed int. | | | x | | x | x |
| alpha | decimal | | | x | | | x |
| alpha | signed dec. | | | x | | x | x |
| integer | group | x | | | | | x |
| integer | alpha | x | | | | | x |
| integer | integer | | | x | | | |
| integer | signed int. | | | x | | x | |
| integer | decimal | | | x | | | |
| integer | signed dec. | | | x | | x | |
| signed int. | group | x | | | x | | x |
| signed int. | alpha | x | | | x | | x |
| signed int. | integer | | | x | x | | |
| signed int. | signed int. | | | x | | | |
| signed int. | decimal | | | x | x | | |
| signed int. | signed dec. | | | x | | | |
| decimal | group | x | | | | | |
| decimal | alpha | *error* | | | | | |
| decimal | integer | | | x | | | |
| decimal | signed int. | | | x | | x | |
| decimal | decimal | | | x | | | |
| decimal | signed dec. | | | x | | x | |
| signed dec. | group | x | | | x | | |
| signed dec. | alpha | *error* | | | | | |
| signed dec. | integer | | | x | x | | |
| signed dec. | signed int. | | | x | | | |
| signed dec. | decimal | | | x | x | | |
| signed dec. | signed dec. | | | x | | | |

Table 3-2: DASDL MOVEs (COBOL Conventions)

If an item is moved from the fixed format part to any variable format parts, the contents of that field will be lost in all records not containing the specified variable format parts. A DASDL warning will be given to indicate the potential loss of data.

If an item is moved from any variable format parts to the fixed format part, that item will be set to initial values if specified or to null in all records which did not contain the specified variable format parts.

If variable format parts are eliminated, there can not be any records in the database which contain that variable format part, otherwise a data error exception will occur during reorganization and the reorganization will fail.

## VERSION CHECKING

Each structure and remap has associated with it a version stamp which reflects the latest time that a change was made to the logical description of the database, that would affect any programs which use that structure or remap. Any programs containing descriptions of a structure or remap with a different version stamp will get a version error when the database is opened. A recompilation of the program is required to bring it up to date with the current description of that structure. This recompilation must take place after the successful completion of the reorganization process. The version stamp of a structure or remap is contained in the LIBRARY file which describes it; the LIBRARY files are not generated until the successful completion of the reorganization. Note that whenever a structure is deleted, its version stamp is changed.

Some of the changes that are legal with reorganization will require that the version stamps of some of the structures or remaps change. The user should be aware of any changes requiring recompilation and the magnitude of the recompilation effort required, before making any changes to the database. All of the reorganization rules marked with an asterisk (*) under Data Base Redescription Rules will require a version stamp change and the necessary recompilations.

## SPECIAL REORGANIZATION FUNCTIONS

Special reorganization processing against the database may be requested by the user. There are two types of these functions: GENERATE and PURGE. These special functions are requested by including explicit reorganization statements in the DASDL source. These statements may be the only source input to the DASDL compiler or they may be included with a redescription of the database. In the latter case, the explicit reorganization statements must appear after the database redescription.

### GENERATE STATEMENT

During the normal updating of a database, the efficiency of the database may deteriorate both in terms of the amount of I/O required to access parts of the database and the amount of wasted disk space.

The GENERATE statement causes the reorganization programs to rebuild structures in the database, restructuring them to increase their efficiency and "garbage collecting" excess disk space.

All structures return unused disk space to their available storage list. However, there is no mechanism for returning unused file areas to the system. Thus, if a structure at sometime included a very large number of records and subsequently returned to a more typical size, none of its unused physical areas would be returned to the system. The space would be available to that structure but may never be needed. A GENERATE on a structure causes the reorganization process to "compress" the structure and return unused file areas to the system.

A GENERATE on a structure, besides "garbage collecting" unused space, causes the reorganization process to rebuild the structure, restoring it to a more efficient state. The specific effect on the structure and improvement in its efficiency is dependent on the structure type.

Syntax:

```
GENERATE -- <structure name> ------------------------------------->#
                               I                                 I
                               I_ (FAMILYNAME = <reorg-pack-id>)_I
```

Semantics:

A GENERATE on a data set causes the reorganization process to read the data records in their physical order and store them into the new database. This also causes an implicit GENERATE on all sets or subsets which reference the data set.

A GENERATE statement on a manual subset causes the reorganization process to do a find for each entry in the old database and an insert for each entry in the new database. In this way, manual subset entries which point at dead records or at records whose keys have changed will be eliminated from the database. However, an implicit GENERATE only causes a balance to be done on the manual subset, and the addresses of the object data records to be adjusted. In this case, those entries which point at dead records are eliminated, but no check is made for matching key values.

A GENERATE on an automatic set or subset causes the reorganization process to perform a balancing operation on the structure. The old structure is read by the reorganization programs and the tables are restructured according to the SPLITFACTOR and MAXENTRIES attributes. If the object data set is also being reorganized, the addresses of the object data records are also adjusted.

A GENERATE on an embedded data set or manual subset will cause an implicit generate on the parent data set, with the exception of an implicit GENERATE on a manual subset which only causes the manual subset's structure heads to be adjusted in the parent data set records.

A GENERATE on an embedded structure will cause groups of records with the same parent record to occupy contigious blocks.

The optional FAMILYNAME clause allows the user to specify an alternate media for the GENERATEd structure. All intermediate or temporary files used in the reorganization process for this structure will reside on the pack specified. See the section on FILE NAMING CONVENTIONS for more details.

Syntax:

```
GENERATE --- <disjoint data set name> ----------------->>

>>-- ORDERED BY -- <ordered set> ---------------------------------->#
                                  |                                |
                                  |_(FAMILYNAME = <reorg-pack-id>)_|
```

Semantics:

This variant of the GENERATE statement is the same in all regards except that the data records will be read in the order of the ordered set rather than in the data set's physical order. The ordered set must exist in the old database.

## PURGE STATEMENT

The PURGE statement is the means by which the user may request that the reorganization process remove all records from a data set, break all relationships that have been established for a manual subset or recreate an automatic set or subset. A PURGE may be thought of as a deletion and addition of a structure where the newly added structure has the same structure number and version stamp as the deleted structure. It should be noted that a PURGE takes precedence over all other reorganization functions.

Syntax:

```
PURGE ----------- <structure name> -------------  ;  -------->#
```

Semantics:

A PURGE of a structure causes its file to be reinitialized and all information to be removed from it. When an embedded structure is PURGEd and its parent structure is not PURGEd then the embedded structure's structure head is set to null in the parent data record.

A PURGE of a data set causes an implicit purge of all its embedded structures and all sets and subsets which reference it.

When an automatic set or subset is PURGEd and its object data set is not PURGEd, the reorganaization process rebuilds it from the data in the object data set. This is different from a GENERATE, in that the GENERATE causes the reorganization process to rebuild the set from itself.

# FILE-NAMING CONVENTIONS

The reorganization programs generate a number of temporary disk files that are used during the reorgaization of a database. The user should avoid naming his files in such a way as to conflict with the names of these temporary files.

A temporary copy of the database dictionary has the name:

```
<database pack>/#<new database name>/DICTIONARY.
```

The <database pack> and <new database name> come from the comoile card of the DASDL compilation.

If the REORG.READER and REORG.WRITER are run simultaneously, they communicate via two queue files:

```
<old database name>/REORGQ  and

<old database name>/ERRQ
```

There are five different types of temoorary files that may be used for a structure in the data base:

1) <reorg pack id>/#<new database name>/#REORG:<structure number>

This file is used for structures which are rebuilt by the system.

2) <reorg pack id>/#<new database name>/#FIXUP:<structure number>

This file is used for structures which are rebuilt by the reorganization routines.

3) <reorg pack id>/#<new database name>/#A-XRF:<structure number>

This file is an address cross-reference for data sets which are reorganized.

4) <reorg pack id>/#<new database name>/#XAREA:<structure number>

This file is used to facilitate changing the number of areas of a structure.

5) <reorg pack id>/#<new database name>/#SORT:<structure number>

This file is used to sort the keys of an index random structure which is being converted from the old table format to the new table format.

The <reorg pack-id> is taken from the FAMILYNAME clause on the GENERATE statement. If no <reorg pack-id> has been specified for this structure then the structure's pack-id in the new database will be used.

When the REORG.READER and REORG.WRITER are run serially, disjoint data hierarchies are passed from the REORG.READER to the REORG.WRITER via a tape file. These files have the name:

<old database name>/#DATAQ:<structure number>

where the structure number is that of the disjoint data set.

All temporary files are removed automatically by the REORG.WRITER at the end of the reorganization process, along with all files in the old database which are not part of the new database.

## ABNORMAL CONDITIONS

Certain abnormal conditions may cause the reorganization to terminate prematurely. When this occurs the user needs to ascertain if the reorganization process is restartable and if not, what action must be taken to restore the existing database to its state prior to initiating the reorganization.

The termination of the reorganization may be initiated externally (program DSed, Clear/Start, etc.) or internally. When internally initiated, the reorganization programs will notify the user whether or not they may be restarted. In general, when the termination is externally initiated, the reorganization is restartable.

The reorganization process has two phases. In the first phase, no modification is made to the existing database. In the second phase, the reorganization will remove, modify, and add files. If the reorganization terminates in this second phase and the reorganization is not restartable, the user should reload his backup copy of the database before rerunning the reorganization or continuing processing on the existing database. The user can identify which phase the reorganization was in by inspecting the output from the reorganization programs. If that output is not available (when the reorganization terminated because of a Clear/Start) the SPO output maybe inspected for the message:

```
"**** BEGIN FILE MODFICATION ****"
```

This message indicates that the reorganization process has begun the second phase.

A certain class of nonrestartable errors occurs because of an improper specification of the database to DASDL. Possible errors of this type are:

-- duplicates which occurred but were not specified as allowed in the new database.

-- a limit error on a file in the new database (e.g., maximum level of coarse tables exceeded, file size exceeded).

-- a data error because of failure to meet WHERE or VERIFY conditions specified in the new database.

-- attempts to reduce the number of areas in a file below the maximum area number that has space allocated (occurs when only the number of areas is changed on a file).

If the output from the reorganization process indicates that it terminated because of one of the errors listed above, the user must correct his DASDL specifications before rerunning the reorganization.

## ALGORITHMS

The DMS subsystem of the MCP is used for much of the reorganization processing. However, there are several special algorithms implemented in the reorganization programs to provide functions that are not available in the DMS subsystem, or to provide speed benefits for frequently used functions. The details of these algorithms are explained below.

Indexed sequential is one of the special functions. The algorithms used in the DMS subsystem attempt to build and dynamically maintain a balanced tree structure; however, additions and deletions tend to unbalance the tree structure. The balancing function reestablishes a balanced tree. It uses as input, the tables built by the DMS subsystem.

Each entry is loaded in order into the fine table until the fine table is splitfactor full. The key of the last entry, together with the address of this fine table, is then propogated up to the first level coarse table. The coarse table is also filled to splitfactor, and when that size is reached the last key and the table address is propogated up to the next level coarse table. New fine and coarse tables are allocated as necessary. When all entries are loaded into the fine tables, the last key of the last fine table (all bits on) is propogated up through all existing levels of tables. The root table pointer is set to the highest level coarse table. During the process of filling tables, each level of tables are linked together in logical order by a next and prior pointer.

Manual subsets are treated as a special case. If a data set to which a manual subset refers is reorganized, then the addresses of the manual subset must be corrected to reflect the new locations of the data records. Garbage-collection of the list structure is performed while correcting the addresses.

The parent records of the manual subset are accessed in physical sequence. As each is accessed, its entire manual subset is processed. First, there is a check for a fast subset (manual subset with one element and no key). If one is found, then the address which is kept in the list head of the parent is corrected and the algorithm moves on to the next parent. Otherwise, the list tables are retrieved in a sequence determined by the list control information of the list tables. Each entry is taken from the old list, its address is corrected, then it is entered into the new list. The new list is compacted and all space for deleted entries in the old list are removed by combining the entries from multiple old tables, until a new table is filled. The list head in the parent record is adjusted to reflect the new list. Because the new list is built starting from the parent record, the new list will have all the tables from a parent contiguous, to take advantage of blocking and to minimize arm movement.

Each time an address correction is made, the address is checked to ensure that the referenced record is still valid. If the old address points at deleted records, then those list elements are not entered into the new list. This process results in new files for both the parent structure and the manual subset.

If a manual subset is reorganized, then the DMS access routines within the MCP are used to read the old list and build a new list.

If a new set or automatic subset is added to an existing data set, then the DMS access routines in the MCP are used to build the new index. First, the structure records are adjusted to indicate that only the new index is to be built, then the reorganization uses the MCP access routines to rebuild the set. At the completion of the operation, the structure records are corrected.

Another special function is the routines which convert pre-8.0 index sequential and index random structures to the 8.0 format. These routines cannot be called directly by the user. They are used by DASDL when the user requests a conversion of a pre-8.0 database to 8.0. The conversion process requires a reorganization, however the change to the new format indexes is the only process which will occur during this reorganization. The user may not request any other reorganization during this execution.

# INDEX

# DOCUMENT TRANSMITTAL

PAGE ____ OF ____

CONTROL DOC.

SYSTEM B1800/B1700

PL FILE NO. ____

PROJ. NO. ____ COORD. ____ CHKR. ____ TO RECORDS ____ DISTRIBUTION ____

| DOC PFX | DOCUMENT NUMBER | REV | UNIT NUMBER | DL NO. | PL FILE NO. | ENG. RECORDS ONLY | | | | | | REMARKS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | QTY REV PAGE | COPIES EACH PAGE | SEPIAS | MT REQ'D | CD OK REQ'D | | |
| P.S. | 2219 0540 | B | | | | 26 | ✓ | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |

DOCUMENT TRANSMITTAL