AAGAARD

PLATANVEJ

Title:

RC3600 DATA ENTRY RELEASE 2 SUPERVISOR PROGRAMMING GUIDE

A/S REGNECENTRALEN

RC SYSTEM LIBRARY: FALKONERALLE 1 DK-2000 COPENHAGEN F

RCSL No:	43-GL 4351
Edition:	June 1977
Author:	Birte Steckhahn

CONTENTS PAGE

CONTENTS (continued) PAGE

1. INTRODUCTION

This manual describes how to make a supervisor program to the Data Entry System, Release 2, and explains the environment where the programs are operated.

The Supervisor is a part of RC3600 Data Entry System, called NANNY which handles loading of and communication with supervisor programs and dialog with the operator, when the key station is in supervisor mode. When it receives a command it looks for the corresponding entry in the RC3600 file system catalog, loads the program to the supervisor area in memory, starts the program as a process and sends a message to the process containing the command line typed by the operator.

Before starting to make a supervisor program, it is recommended to glance through the following sections :

2., 3., 5.1, 5.2, 5.3, and 6.

2. DISC FILES

2.1 Libraries

The Data Entry System has 5 libraries. They are physically disc files containing a group of names.

The 5 libraries are :

1. Job Library

   The library contains the names of all the jobs in the system. The name of the file on the disc is JBLIB. The structure of the library, see appendix A.

2. Format Library

   The library contains the names of all the translated formats in the system. The name of the file on the disc is FBLIB. The structure of the library, see appendix A.

3. Subprogram Library

   The library contains the names of all the translated sub-programs in the system. The name of the file on the disc is SPLIB. The structure of the library, see appendix A.

4. Table Library

   The library contains the names of all the translated tables in the system. The name of the file on the disc is TBLIB. The structure of the library, see appendix A.

5. Disc-table Library

   The library contains the names of all the created disc-tables in the system. The name of the file on the disc is DTLIB. The structure of the library, see appendix A.

2.2 Jobs

A job is a disc file containing a group of batch names, and for each batch name a status word containing a status of the batch.

Each job name is included in the job library (JBLIB), and in each batch belonging to a job is the job name mentioned, too. Further description of a job file, see appendix B.

2.3 Batches

A batch is a disc file on the disc containing the data and register records together with some informations about the batch, such as which job it belongs to. Every record then contains informations about itself, such as a record status, and a number of fields.

The informations about the batch is placed in the first block in the file and this block is called the batch head. The information about a record is positioned at the beginning of the record and is called the record head.

A batch is terminated by a batch end mark.

The exactly structure of a batch, see appendix C.

3. CODING OF A SUPERVISOR PROGRAM

All supervisor programs must be coded in the MUSIL programming language and compiled using the MUSIL compiler. Output from the MUSIL compiler is a relocatable binary object code, which can be added to the Data Entry System by loading it to disc (see section 7).

The standard code procedures described in section 4 have been implemented in order to make it possible for the Data Entry Supervisor to

- transfer command lines and parameters to the supervisor programs, and to get receipts from the programs before removal of the process after program termination. (The procedures 'get command', 'get parameter', and 'return').

- lookup, delete or insert a name in a library or to seach sequential through a library (the procedures 'find item', and 'get next item').

- read the fields in a batch (the procedures 'access' and 'allaccess').

- slow down the supervisor program so that keying, rekeying, and editing will be unaffected of the run of a supervisor program (the procedure 'delay').

- to open a file for the printout (the text that must be written on the hard copy device or on the supervisor key station display) (the procedure 'connect file').

These procedures must be declared and called in the MUSIL program and copied into the program at compilation time. See reference 1.

4. DESCRIPTION OF STANDARD CODE PROCEDURES

4.1 Code Procedure Get Command

This procedure is used to receive a command line from the supervisor in the Data Entry System.

The call of this procedure must be the first statement executed in the supervisor program.

The procedure must be declared as follows:

```
procedure cmmd (var comline: string (112);
                var return:  integer);
codebody;
```

Parameters:

Comline: Return parameter of type 'string'.

The length of the parameter must be at least 112 bytes.

The parameter contains a command line, typed in by the keying operator, and must not be changed by the program.

Return: Return parameter of type 'integer'.

The parameter contains the message address used by the supervisor.

This parameter must be used when returning to the supervisor (see section 4.3), and when delaying the program (see section 4.9), and the value of the parameter must therefore not be changed by the program.

After a syntax check, made by the supervisor, each parameter to the program will be packed in 'comline' as groups of information as follows:

1. Type of parameter

   2 bytes; where 0 = integer parameter
   1 = text parameter
   2 = termination

2. The parameter

   2 bytes for integer parameters, and
   6 bytes for text parameters, each parameter terminated by at least one null character

3. Terminator for parameter

   2 bytes, where 0 = space
   1 = period
   2 = termination

After a call of the procedure return (see section 4.3) it is possible to get an empty command line (i.e. only ENTER has been pressed on the supervisor key station). The contents of 'comline' will then be: <0> <255> if no syntax check has been made, or <0> <2> after a syntax check (i.e. termination).

4.2 Code Procedure Get Parameter

This procedure is used to get a single parameter from the command line. The first call of this procedure in the program will in parameter 'item' return the name of the file that later on must be opened for printout (see section 5.1) with the procedure 'connectfile' (see section 4.6). The parameter 'value' describes if the printout is to be produced on a hardcopy device or on the supervisor key station display screen. The contents of those two parameters must therefore be saved before a new call of the procedure 'get parameter'. The subsequent calls will return the parameters in the order in which they were typed on supervisor key station.

The procedure must be declared as follows:

```
procedure gtpm (var comline: string (112);
                var item:    string (6);
                var value:   integer;
                var kind:    integer;
                var sep:     integer);
codebody;
```

<u>Parameters</u>:

Comline: Call parameter of type 'string'. See section 4.1.

Item: Return parameter of type 'string'.

The length of the parameter must be a least 6 bytes.

The parameter contains a text from the command line if kind = 1.

The text is terminated by a least one null character.

The first character in a text parameter must be a letter followed by letters or digits.

The first time the procedure is called, this parameter contains the name of the printout file.

Value: Return parameter of type 'integer'.

The parameter contains an integer value if kind = 0.

An integer parameter may be typed on the supervisor key station either in signed/unsigned decimal representation, or in octal representation (identified by a preceeding apostrophe). The first time the procedure is called, this parameter indicates whether the printout must be produced on a hard copy device or on the supervisor key station display screen:

Value = 0: printout on display screen.

Value = 1: printout on hard copy.

Kind:	Return parameter of type 'integer'. The parameter contains the type of the returned parameter, where: Kind = 0 means integer parameter. Kind = 1 means text parameter. Kind = 2 means that the procedure has been called too many times (i.e. more than the number of parameters in the command line), or that an empty command line has been received (i.e. only ENTER has been pressed on the supervisor key station) after a call of the procedure return (see section 4.3).
Sep:	Return parameter of type 'integer'. The parameter contains an integer value which indicates the terminator of the returned parameter, where: Sep = 0 means space Sep = 1 means period Sep = 2 means termination = last parameter (i.e. ENTER on the supervisor key station).

4.3 Code Procedure Return

This procedure is used to return to the Supervisor in the Data Entry System, either to get a new command line from the supervisor key station or to indicate that the program execution must be terminated and the process must be removed from the supervisor area in memory.

In case of termination the call of this procedure must be the last statement executed in the supervisor program.

The procedure must be declared as follows:

```
procedure return (var return:    integer;
                  var result:    integer;
                  var action:    integer;
                  var textmode:  integer;
                  var text:      string (80 or 160));
codebody;
```

Parameters:

Return:	Call parameter of type 'integer'. See section 4.1.
Result:	Call parameter of type 'integer'. This parameter may contain a number, which may be output on the supervisor key station display screen as an octal number, when a standard text is output (see parameter 'action'). Standard texts, see appendix E. Result cannot be output after a text in the parameter 'text'.
Action:	Call parameter of type 'integer'. The parameter contains information about a possible standard text to be output on the supervisor key station display screen, special actions to be taken by the supervisor, and about continuation of the program execution. The information must be packed as follows:

(txt1 shift 10) + (txt2 shift 4) + (check shift 3) +
(printout shift 2) + (outputres shift 1) + cont.

txt1: If a standard text must be output, the number of the first text must be placed here.
txt1 = 0 means that no text is output, a text in the parameter 'text' is output or a semaphore function has to be performed.

txt2: If a standard text is composed of two text numbers the second text number must be placed here. If a semaphore function has to be performed, the ident of the semaphore must be placed here (first semaphore = 1). 'txt2' = 0 means that only one standard text number is used (indicated in 'txt1'), no text is output, or that a text in the parameter 'text' is output.

check: = 1 if the program execution must continue ('cont' = 1) and the new input from the supervisor must be syntaxchecked or if a semaphore function has to be performed.
= 0 if no syntax-check of new input or if the program has to terminate ('cont' = 0).

printout: = 1 if

- a printout is produced and must be output to the hard copy device or the supervisor key station display screen.
- a signal semaphore function has to be performed.

= 0 if

- a printout must not be output (for example after an error) or a printout has not been produced.
- a wait semaphore function has to be performed.

outputres: = 1 means that the number in the parameter 'result' must be output on the display screen after a standard text.
= 0 means that the parameter 'result' must not be output after a standard text, or that a text in parameter 'text' is to be output ('result' cannot be output after such a text). In this case the contents of 'result' are irrelevant.

cont: = 1 means that program execution must continue.
= 0 means program termination.

Textmode: Call parameter of type 'integer'.

This parameter describes the contents of parameter 'text'.

= 0 means that the contents of 'text' are irrelevant, because the parameter 'action' is used for receipt specification.

= 1 means that byte 1-80 of parameter 'text' contains a text to be output on the supervisor key station display screen instead of a standard text.

= 2 means that byte 1-80 of 'text' contains a replace command (see 'text').

= 3 means that byte 1-80 of 'text' contains a text to be output and byte 81-160 of 'text' contains a replace command.

Text: Call parameter of type 'string'.

The length of the string must be 80 or 160 bytes, depending on the value of 'textmode'.

Both a text to be output on the supervisor key station display screen and a replace command must be terminated by a null-character.

A replace command is a supervisor command, i.e. a call of another supervisor program with termination of the first program in the same way as if the first program terminates, and the operator calls the second program.

A replace command cannot change the output device for the printout. If a program is called with output of printout on printer via spool-file, and this program terminates with a replace command, the new program will place the printout on the printer via spool-file, too.

In appendix D a diagram is showing how to use the procedure.

4.4 Code Procedure Access

The procedure is used to read transfer fields in the data records in a batch. The zone connected to the batch must be opened for reading and positioned just after the block with the batch head, i.e.

```
zone.zname:= <batchname>;
open (zone, 1);
'. read and check the batchhead'.
setposition (zone, 0, 1);
```

Between the first and the last call of the procedure the zone may not be changed by calling other procedures using the zone.

The procedure will skip register records and the fields with status 'no transfer' (register records and field states are described in appendix C).

The procedure must be declared as follows:

```
procedure access (file          z;
                  var field:    string (80);
                  var number:   integer;
                  var status:   integer;
                  var subname:  string (2));
codebody;
```

<u>Parameters</u>:

z:	Call and return parameter of type 'file'. The parameter must be defined as a disc-file, i.e. kind = 62 and share length = 512. z. zname must be a name of a batch, e.g. <u>not</u> a name of a job or a table.
Field:	Return parameter of type 'string'. The length of the string must be 80 bytes. The contents of 'field' is a field with the status transfer, if the parameter 'number' is greater than or equal to zero. 'Number' then indicates number of characters in the field.
Number:	Return parameter of type 'integer'. The parameter describes what has been read: ≥ 0: a field has been read. 'Number' contains the number of characters in the field which is placed in 'field'. The parameter 'status' contains the field status and the parameter 'subname' is undefined. $= -1$: a record head is read. 'Field' is undefined, 'status' contains the record status and 'subname' contains the name of the subformat, which has been used when keying the record.

	=-2:	a record end is read. 'Field', 'status' and 'subname' are all undefined.
	=-3:	a batch end mark is read. 'Field', 'status' and 'subname' are all undefined.

Status: Return parameter of type 'integer'.

The parameter contains the record status if a record head is read and the field status if a field is read. Otherwise 'status' is undefined.

Subname: Return parameter of type 'string'.

The length of the string must be 2 bytes.

If a record head is read the parameter in the first byte contains the name of the subformat, which has been used when keying the record. The second byte is a null-character.

4.5 Code Procedure Allaccess

This procedure is used in the same ways as the procedure 'access', and returns nearly the same. The only differences between 'access' and 'allaccess' are

- 'allaccess' is defined with the name allac instead of access.
- 'allaccess' returns fields with status no transfer, which are ignored by 'access'.

The procedure is defined:

```
procedure allac (file              z;
                 var field:        string (80);
                 var number:       integer;
                 var status:       integer;
                 var subname:      string (2));
codebody;
```

<u>Parameters</u>:

Please consult section 4.4.

4.6 Code Procedure Connect File

This procedure is used for opening the file where the printout must be placed (a work-file on the disc or one of the hardcopy devices).

The procedure looks up the catalog entry given by the name. If the entry is found and is a file descriptor (if the name is one of the names of hardcopy devices), the information about device name, mode, kind, file, block, and give up mask are transferred to the zone variables.

If the name is the name of the work-file, a 'createentry' is performed (because the entry is deleted by the supervisor before calling the program).

Before return to the MUSIL program the zone is opened with the mode specified.

The procedure must only be called once in a program.

The procedure must be declared as follows:

```
procedure connec (file              z;
                  const outmode:    integer;
                  const name:       string (6));
codebody;
```

Parameters:

z:	Call and return parameter of type 'file'. The parameter must be defined as a file with the sharelength = 512.
Outmode:	Call parameter of type 'integer'. The parameter must be set to output mode, i.e. `outmode:= 3;` before call of the procedure.
Name:	Call parameter of type 'string'. The length of the string must be 6 bytes. The contents of 'name' must be the name of the file where the printout must be placed, (this name is received when the procedure 'get parameter' (section 4.2) is called the first time).

4.7 Code Procedure Find Item

This procedure is used to search for a name, to insert a name or to delete a name in a library or a job.

The procedure must be declared as follows:

```
    procedure fitem (file            z;
                     const optype:   integer;
                     const length:   integer;
                     const name:     string (6));
    codebody;
```

NOTICE:

In the MUSIL program there must be two integers defined just after the string 'name', for an extraword and a result, because the procedure uses those two integers, as they were parameters:

If z. zname is a name of a job, i.e. 'name' is a name of a batch (specified by 'length' = 4, see 'length') the status of the batch taken from the job is placed in the first integer, called 'extraword', and the result of the call of the procedure is placed in the second integer, called 'result'.

Parameters:

z: Call parameter of type 'file'.

The parameter must be defined as a disc-file, i.e. kind = 62 and sharelength = 512.

z. zname must be the name of the library or the job, where the procedure must look for 'name', insert it or delete it. The zone must be opened for reading before a call of the procedure.

Optype: Call parameter of type 'integer'.

The procedure first searches through the library or the job for 'name' and then looks at 'optype' to decide the action to be taken :

=0: search item:
If 'name' is found, 'result' is set to 0, otherwise to 1.
If 'name' is a batch name the status from the job is placed in 'extraword'.

= 1 : delete item:
If 'name' is found, the name is deleted and 'result' is set to 0, otherwise to 1.

	=2:	insert item: If 'name' is not found, the name is inserted and if 'name' is a name of a batch, the contents of 'extraword' is inserted after 'name' and 'result' is set to 0. If 'name' already exists, 'result' is set to 1.

Length: Call parameter of type 'integer'.

This parameter describes how many words in the library or the job, the item must use. If z. zname is a name of a job, 'length' must be 4 (3 words for the name and 1 word for the status). If z. zname is a name of a library (job, format, subprogram, table, or disc table library), 'length' must be 3.

Name: Call parameter of type 'string'.

The length of the string must be 6 bytes, and the contents must be the name to search, insert or delete in the job or the library. A name must be a letter followed by not more than 4 letters or digits and terminated by at least one null-character.

Extraword: This is not a parameter used in the call of the procedure, but is an integer defined just after 'name' in the variable section in the MUSIL program.

If 'length' = 4 the batch status from the job is placed here if 'optype' = 0, and if 'optype' = 2 the contents of 'extraword' is inserted after 'name'.

Result: This is not a parameter used in the call of the procedure, but is an integer defined as second integer after 'name'.

The contents of 'result' is 0 if the call of the procedure causes no errors, otherwise 'result' is set to 1.

4.8 Code Procedure Get Next Item

This procedure is used to run sequential through a job or a library to get all the names one by one, e.g. run through a job in order to dump all the batches.

The zone with the job or library name must be opened for reading and positioned on the first block before the first call of the procedure.

The procedure must be declared as follows:

```
procedure getnex (file              z;
                  const length:     integer;
                  var name:         string (6);
                  var extraword:    integer;
                  var status:       integer);
codebody;
```

Parameters:

z:	Call and return parameter of type 'file'. The parameter must be defined as a disc-file, i.e. kind = 62 and sharelength = 512. z. zname must be the name of the job or the library containing the names to be read.
Length:	Call parameter of type 'integer'. This parameter describes how many words in the library or the job one item uses. If z. zname is a name of a job, 'length' must be 4, and if z. zname is a name of library, 'length' must be 3.
Name:	Return parameter of type 'string'. The length of the string must be 6 bytes. The contents of 'name' after a call of the procedure is the next name read in the job or the library.

Extraword: Return parameter of type 'integer'.

If z. zname is a name of a job, the batch status from the job is placed here. If z. zname is a library name it is undefined.

Status: Call and return parameter of type 'integer'.

Before the first call of the procedure, 'status' may be undefined, but before the second and all the other calls it must have one of the following values:

= 0: The disc zone has been used for something else, i.e. the zone has been destroyed and the job or library name must be inserted in z. zname before calling the procedure.

= 1: The disc zone has not been used after last call.

= -1: This value must be used in the first call with a new job or library name in z. zname after the procedure has been used on one job or library.

When the procedure returns to the MUSIL program, 'status' has the values:

= 0: The end of the job or library has not been reached yet.

= 1: The end of the job or library has been reached. 'Name' and 'extraword' are undefined.

4.9 Code Procedure Delay

This procedure is used to slow down a supervisor program with many disc transports in order to let keying operators work without speed-reduction, so the procedure must be used in programs, that read or write on the disc.

The procedure makes the MUSIL program wait a number of timer-periods (one timer-period = 20 milliseconds) standing in mess3 in a message buffer. The number is calculated by NANNY and depends on number of key stations in key, rekey and edit mode. The procedure is only effective if the supervisor key station has come to the supervisor mode by the control command SUPERVISOR SLOW. If the control command SUPERVISOR is used, mess3 is always set to 0.

The procedure is declared as follows:

```
procedure delay (var delayed: integer;
                 var bufaddr: integer);
codebody;
```

Parameters:

Delayed: Return parameter of type 'integer'.

After a call of the procedure 'delayed' will contain the number of timer-periods the program has waited.

Bufaddr: Call parameter of type 'integer'.

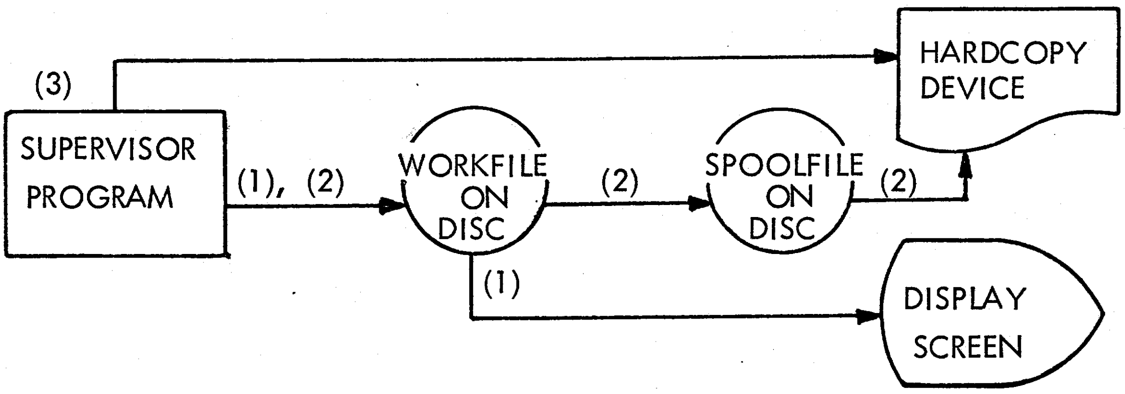
The parameter must contain the address of that message buffer, where mess3 contains the number of timer-periods, the calling program must wait. This address is the return-value of the parameter 'return' in the code procedure 'get command' (see section 4.1).

5. PROGRAMMING HINTS

5.1 Printout

A supervisor program can produce a printout in the form of a survey, log, listing, and the like. This section describes the administration of the printouts and how to make a printout.

The printouts can be output in three ways :



(1) On the supervisor key station display screen.

(2) On a hardcopy device (e.g. line printer or operator console device) via a so-called spool-file.

(3) Directly on a hardcopy device.

The way printout is to be produced is selected by the operator when the supervisor command is entered.

Examples:

SYSTEM DISC	Printout on display screen.
SYSTEM.S DISC	Printout on hardcopy device via spool-file.
SYSTEM.L DISC	Printout directly on hardcopy device.

5.1.1 Spool-File

The spool-file can contain printouts from several supervisor programs and the operator can stop and start printing from the spool-file. This means that the hardcopy device may be used for non-data entry functions (for example conversion) simultanously with production of some data entry printouts to be printed later.

5.1.2 Hardcopy Devices

When a system-tape is installed some catalog entries are created to describe hardcopy devices. It can be such as the line printer or the operator console device (TTY).

The entries contain information about the device: device name, mode, kind, file, block, and giveup-mask.

The names of the entries are:

HCOPY	Normal hardcopy device. It depends on the actual configuration, but can be the line printer for instance.
HLPT	Line printer
HLPT1	Second line printer
HSP	Serial printer
HSP1	Second serial printer
HCPT	Charaband printer
HCPT1	Second charaband printer
HTTY	Operator console device (TTY)

The current hardcopy device can be changed to another one by the SPOOL NEWDEVICE - command (see reference 2, section 9).

5.1.3 Opening of Actual Device

When a supervisor program is called, the Supervisor finds out, where the printout must be placed. If it is in the work-file, this file is deleted and later on created again (by code procedure 'connect file'). When the code procedure 'get parameter' (section 4.2) is called the first time the name of the work-file or the name of the catalog entry describing current hardcopy device is returned in the parameter 'item'. In parameter 'value' the procedure returns if the printout must be written on the display screen or on a hardcopy device:

value = 0 :	display screen
value = 1 :	hardcopy device.

It is necessary to know the length of the lines to be written because the display screen has space for 80 characters, and a line printer has space for 132 characters.

To open the file the code procedure 'connect file' is used (see section 4.6). This procedure looks up the name in the catalog. If the entry is found, it is one of the names mentioned in section 5.1.2, and the procedure moves the device name, kind, and giveup-mask to the zone and or's the mode found with the mode given in the call of the procedure. If the entry is not found, it is the name of the work-file, which was deleted when calling the supervisor program. The procedure then creates an entry as a disc file. Hence it is very important that this procedure is called only once in a program. Finally the procedure opens the file and positions it at file and block.

5.1.4 How to make the Printout

A text string is written in the printout file by using the standard procedure OUTTEXT. This procedure stops writing the text string in the zone, when it meets a null-character, i.e. all strings to be used by OUTTEXT must be terminated by at least one null-character. The character set to be used must be the ASCII code. When the printout is output to the actual device it is automatically converted if the output device does not use the ASCII code (for example a line printer). The last character to be output in the prinout before closing the zone must be an end medium character: value 25.

The last statement to be executed in a supervisor program must be a call of the code procedure 'return' (section 4.3) with the parameter 'cont' equal to 0. If the program has made a printout with the end medium character as the last one, the parameter 'printout' must be set to one. This informs the Supervisor that a printout has been made, and if it is written in the work-file, the Supervisor copies the printout from the work-file to the display screen or to the spool-file and from the spool-file to the hardcopy device. The copying is stopped when the end medium character is met. If the printout is copied to the display screen and a line contains more than 80 characters, the Supervisor automatically inserts a 'new line'-character after the first 80 characters.

All control characters (i.e. value less then 32 = SP) are ignored except 'new line' characters, this means that 'form feed' characters, for instance, has no effect.

5.2 Program Termination

Before the program returns to the Supervisor by a call of the code procedure 'return' with the parameter 'cont' = 0, all processes (i.e. driver- and area processes) which have been used by the program must be released. This is done by calling the procedure CLOSE with non-zero release (i.e. close (zone, 1);). Before closing the printout file, an end medium character must be OUTTEXTed. When using the code procedure 'return' it is important to set the values of the parameters in the right manner.

Parameter 'action':

'cont'	must be zero
'check'	must be zero
'printout'	must be 1 if a printout with an end medium character has been mode, otherwise to zero.
'outputres'	must be set to 1 if 'txt1' and 'txt2' describe standard texts and if the contents of parameter 'result' must be output on the display screen after the standard texts. Otherwise 'outputres' must be set to zero.

Parameter 'textmode':

If this parameter is not initialized in the right way, it may causes some error-messages from the Supervisor.

5.3 Error Procedures

After an error which cannot be corrected by the operator or no correction is wanted, the program execution must be terminated as described in section 5.2. When the procedure CLOSE is called a program loop may occur after an output error because closing output zone may result in a new output message which will call the giveup-procedure again and so on.

This program loop can be avoided in two ways:

1 : The mode in the erroneous zone is set to zero before closing the zone. This will prevent further output messages and the procedure CLOSE will only release the process, but some output data may be lost. (This action is shown in the program example in appendix F).

2 : An error-counter is used to count number of errors. When a zone is opened the counter is set to zero. If the giveup-procedure is called the counter is increased by one and the program tries to close the zone. This may result in one of two things:

- the giveup-procedure is called once more and the counter is increased by one
- the error is corrected and the zone will be closed.

If the error counter then gets greater than for example N the CLOSE-call cannot succeed and it is then skipped.

If a call of CLOSE is succeeded the error-counter is then set to N. (This action is shown in the program example in appendix G, with N = 4).

5.4 Creation of a Batch

If a batch is created by a supervisor program there are different things to remember:
(The structure of a batch, see appendix C)

a) in the batch head:

- jobname must be a name of a job file. If the job does not exist already, the job name must be inserted in the job library (JBLIB) and a file with the job name must be created and the contents must be set according to the specifications given in appendix B.
- the format name must be a name of an existing translated format if the batch later on must be rekeyed or edited.
- the batch name must be the same name as the name of the file where the batch is placed.
- the batch status word must be set to closed, i.e. bit 0 = 1 (status = 100000_8) or to closed and invalid, i.e. bit 0 = 1 and bit 12 = 1 (status = 100010_8). If the invalid bit (bit 12) is set, some of the field in the batch may be invalid.
- number of blocks (word 14) and number of records (word 15) must be calculated and inserted in the batch head.
- end format (word 26) must be set to 100000_8 as if the 'end'-statement in the format was executed.
- the other words in the batch head must be set to zero.

b) in the data record head:

- subname must be the name of a subformat in the format specified in the batch head.
- the length of the record and the record number in the batch must be calculated and inserted.
- the other bytes must be set to zero. The record status can be set to non-zero, if the record must not be dumped. (Normally the supervisor program 'dump' will skip records with invalid status).

c) in the data record end:

- subname must be the same as subname in the record head.
- record end mark must be one byte with the value 28_{10}.

d) in the fields:

- the field status can be set to not transfer if the data in the field must not be dumped or transferred.
- the field end mark must be one byte with the value 27_{10}.

e) register records:

- the register records may <u>not</u> be made by a supervisor program.

If a batch, made by a supervisor program, must be corrected with EDIT the batch must contain register records. They are made in this way:

1) leave supervisor mode with the supervisor command STOP

2) key: EDIT <batchname>

3) key: END

This will causes the editor to run through the batch and make the register records. At the same time the states of the fields, the records and the batch will be set according to the format (specified in the batch head).

After this the batch can be corrected with EDIT.

6. OPERATOR COMMUNICATION

When a supervisor program has been called from the supervisor key station, the communication with the program may be done either from the supervisor key station, if the program uses the code procedures 'get command', 'get parameter' and 'return', or from the RC3600 operator console device (TTY), if the program uses the standard operator communication procedures described in reference 1.

The latter communication is recommended for greater supervisor programs such as print image programs, paper tape conversion programs etc., because it gives a possibility for temporary halts of the program execution caused by the operator.

When using the code procedures 'get command', 'get parameter', and 'return' all the communication between the program and the operator is automatically written in the log-file by the Supervisor. This means that the supervisor program has not to make any action to write in the log-file. (The log-file can be dumped and printed by the standard supervisor programs DUMPSTAT and LISTLOG, see reference 2, section 9).

As can be seen in appendix F communications via the key station has been made possible by

1) calling the codeprocedure 'return' with 'cont' = 1.

2) calling the codeprocedure 'get command', this will wait until the operator has answered the question asked for with 'return'.

   The answer is given in 'comline' as a return parameter from 'get command'

   - if the answer has been syntax-checked by the Supervisor the items from the answer must be picked up by using 'get parameter'.

   - if the answer has not been syntax-checked 'comline' holds a normal text string.

7. INSTALLATION OF NEW SUPERVISOR PROGRAMS

A new supervisor program is put into the Data Entry System by means of the standard supervisor program PUT. In this way it is not necessary to generate a new system tape every time a new supervisor program has been made.

The binary supervisor program (i.e. the output from the MUSIL compiler) may be read from either paper tape or magnetic tape.

The supervisor command for PUT is fully described in reference 2, section 9. Please consult this description.

APPENDIX A: STRUCTURE OF A LIBRARY

The five libraries in the Data Entry System are structured in the same way. Each of them is an extensible disc file containing a group of names with the same characteristics. A disc file is a collection of blocks. The first two bytes in the first block is a counter containing number of names in the library. The first two bytes in the other blocks are undefined. The rest of each block is divided into groups of 6 bytes, each group containing one name. One block can in this way contain 85 names (one block = 512 bytes: two bytes for counter or undefined, 510 bytes for names of 6 bytes = 85 names).

A name consists of a letter followed by not more than 4 letters or digits terminated by at least one null-character.

Examples:

name BAJ01

1. byte	B	A	2. byte
3. byte	J	0	4. byte
5. byte	1	null	6. byte

name B75

1. byte	B	7	2. byte
3. byte	5	null	4. byte
5. byte	null	null	6. byte

When a name is deleted by the code procedure find item (section 4.7) the first two bytes in the name are overwritten with null-characters and the counter is decreased by one. This makes a free space in the library. When a name is inserted in the library by the same procedure, it is inserted on the first free place (and not after all the other names) and the counter is increased by one.

APPENDIX B: STRUCTURE OF A JOB

A job is an extensible disc-file containing a group of batch names and for each batch name a status of the batch.

The first two bytes in the first segment is a counter containing number of batches in the job. The first two bytes in the other blocks are undefined. The rest of each block is divided into groups of 8 bytes, each group containing one batch name and the status of the batch. One block can in this way contain 63 batch names with states (One block = 512 bytes: two bytes for counter or undefined, 510 bytes for names and states of 8 bytes = 63 names, rest 6 bytes).

A name in a job (a batch name) is structured in the same way as a name in a library (see appendix A).

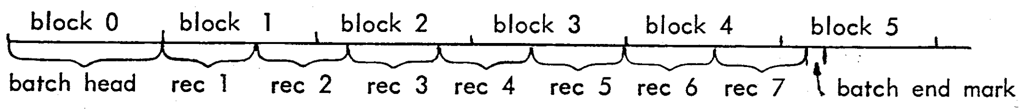
The status of a batch in a job tells, if the batch is used for keying, rekeying or editing:

status = 0:	the batch is not in use.
status = 1:	the batch is used for keying.
status = 2:	the batch is used for rekeying.
status = 3:	the batch is used for keying and rekeying.
status = 4:	the batch is used for editing.

APPENDIX C: STRUCTURE OF A BATCH

A batch is a disc-file containing the records with the fields. The records are stored from the second block and forward in the file. The first block contains some information about the batch and is called the batch head. The batch is terminated by a batch end mark.

Example:



C.1. Batch Head

The contents of the batch head is:

word 0 - 2:	Job name: the name of the job, to which the batch belongs.
word 3 - 5:	Format name: The name of the translated format to be used when keying, rekeying and editing the batch.
word 6 - 8:	Batch name.
word 9:	Batch status (see below).
word 10:	Byte count in old batch last block.
word 11:	Byte count in work batch last block.
word 12:	Block address of last register record.
word 13:	Byte address of last register record.
word 14:	Number of blocks in the batch inclusive the batch head.
word 15:	Number of data records in the batch.
word 16:	Number of rekeyed records.
word 17:	Number of invalid records.
word 18:	Activation time, real time clock 1.
word 19:	Activation time, real time clock 2.
word 20:	Block count in old batch.
word 21:	Block count in work batch.
word 22:	Name of current subformat.
word 23:	Current recordnumber in old batch.

word	24:	Current record number in work batch.
word	25:	Maximum number of records in work batch.
word	26:	End format: = 100000_8 indicates that the 'end'-statement in the format is executed.
WORD	27:	BATCH STATUS2 (see below)

The rest of the batch head is not in use yet, i.e. must all be zeroes.

The batch status2 word is 16 bit, where the value of all the bits, except the last one (bit 15) gives a status:

all bits = 0	: the batch is empty.
bit 0 = 1	: the batch is closed.
bit 1 = 1	: the work batch is closed.
bit 2 = 1	: the batch is editing.
bit 3 = 1	: the batch is rekeying.
bit 4 = 1	: the batch is keying.
bit 5 = 1	: the batch must be rekeyed (the parameter 'rekey' has been used in the 'set'-command).
bit 6 = 1	: the batch has been rekeyed.
bit 7 = 1	: the batch has been partial rekeyed.
bit 8 = 1	: the batch has been edited.
bit 9 = 1	: the batch has been saved by the supervisor program 'save'.
bit 10 = 1	: the batch has been dumped by the supervisor program 'dump'.
bit 11 = 1	: the batch has been transferred.
bit 12 = 0	: the batch is valid.
bit 12 = 1	: the batch is invalid, i.e. one or more fields are invalid.
bit 13 = 1	: the batch has been sorted.
bit 14 = 1	: the batch must be valid (the parameter 'valid' has been used in the 'set'-command).
bit 15	Not used

BATCH STATUS2

b0 = 1	The batch is a copy of another batch
b1 = 1	It is not allowed to key, rekey or edit in this batch
b2 = 1	It is not allowed to key or rekey in this batch
b3-15	Not used

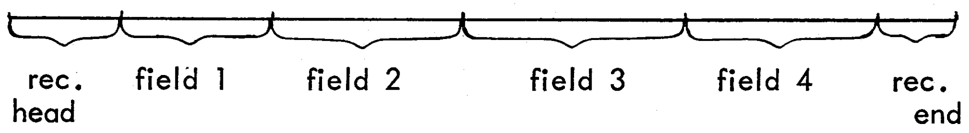
C.2. Records

There are two types of records: data records containing the data, which has been keyed to the batch, and register records, made by NANNY as a copy of the register area, and is written in the batch for every 10 data records.

C.2.1 Data Records

A data record contains a record head, a number of fields and a record end.

Example :



The contents of the record head is:

byte 0:	Subname: the name of the subformat used when keying the record.
byte 1-2:	The total record length.
byte 3-4:	Record number in the batch.
byte 5:	Record status: number of invalid fields in the record.
byte 6:	Rekeyed.
byte 7-8:	Block number of corresponding record in old batch.
byte 9-10:	Byte number of corresponding record in old batch.

The contents of the record end is:

byte 0:	Subname the same as byte 0 in the record head.
byte 1:	Record end mark: one byte with the value 28_{10}.

The contents of a field is:

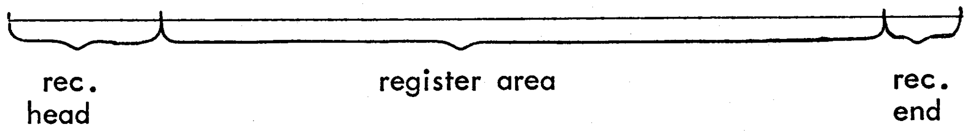
byte 0:	Field status:
	bit 7 = 1 : invalid
1	bit 7 = 0 : valid
2	bit 6 = 1 : skip
4	bit 5 = 1 : skip by statement
	bit 5 = bit 6 = 0 : not skip 8-16 bits 3-4 n.t. unused
128	bit 0 = 0 : transfer field 32 bit 2 : NO CONTENT
	bit 0 = 1 : no transfer field 64 bit 1 = DISPLAY OK
byte 1 → n:	data (max 80 characters in ASCII code with values greater than 31_{10}).
byte n + 1:	field end mark, value 27_{10}.

Fields with the status no transfer and fields with the length 0 are placed as the last fields in the record.

C.2.2 Register Records

A register record contains a record head, a register area and a record end.

Example:



The contents of the record head is:

byte 0:	Subname, the ASCII character '?', value 63.
byte 1-2:	Total record length.
byte 3-4:	Record number in batch.
byte 5-6:	Not used for register records.
byte 7-8:	Block number of previous register record.
byte 9-10:	Byte number of previous register record.
byte 11-12:	Register top.
byte 13-14:	Number of rekeyed records.
byte 15-16:	Number of invalid records.

The contents of the register area is:

word 0 :	Pointer to length, type, and contents of register 01.
word 1 :	Pointer to length, type, and contents of register 02.

And so on to the last pointer to length, type, and contents of last register.

After these pointers each register is described as follows:

byte 0 :	Length of register X
byte 1 :	Type of register X.
byte 2 → n+1:	Contents of register X, n = the length of register X.

The contents of the record end is:

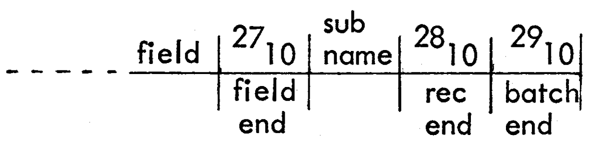
byte 0 :	Subname, the same af byte 0 in the record head, i.e. '?'.
byte 1 :	Record end mark: one byte with the value 28_{10}.

The register records may not be made by a supervisor program. They are made as checkpoints of the current state of the batch after keying a number of data records.

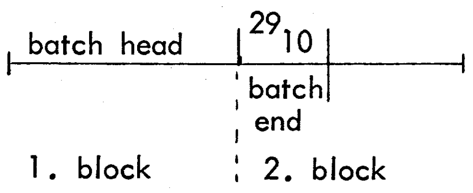
C.3 Batch End Mark

The batch end mark is one byte with the value 29_{10}.

The last bytes in a batch are then:



If the batch is empty the second block of the batch contains a batch end mark in the first byte, i.e.



APPENDIX D: DIAGRAM OF HOW TO USE THE CODE PROCEDURE 'RETURN'

parameters / function	Action: txt 1 bit 0-5	txt 2 bit 6-11	check bit 12	printout bit 13	outputres bit 14	cont bit 15	Result	Textmode	Text
1: No text output get new input	Zero	Zero	C (see below)	O (see below)	Zero	1	irrelevant	Zero	irrelevant
2: Standard text output get new input	first textno.	second textno.	C (see below)	O (see below)	P (see below)	1	no. to be output	Zero	irrelevant
3: Param. 'text' output get new input	Zero	Zero	C (see below)	O (see below)	Zero	1	irrelevant	1	byte 1-80 = text to output
4: No text output termination	Zero	Zero	Zero	O (see below)	Zero	Zero	irrelevant	Zero	irrelevant
5: Standard text output termination	first textno.	second textno.	Zero	O (see below)	P (see below)	Zero	no. to be output	Zero	irrelevant
6: Param. 'text' output termination	Zero	Zero	Zero	O (see below)	Zero	Zero	irrelevant	1	byte 1-80 = text to output
7: Replace command termination	R1 (see below)	R1 (see below)	Zero	O (see below)	Zero	Zero	R1 (see below)	R2 (see below)	R2 (see below)
8: Wait / signal semaphore	Zero	semaph. ident.	1	wait = 0 signal = 1	Zero	Zero	irrelevant	Zero	irrelevant

C :	The new input must be syntax checked:
	check = 0 : yes
	check = 1 : no

O :	A printout has been produced and must be copied to hardcopy device via spoolfile or to supervisor key station display screen.
	printout = 0 : no
	printout = 1 : yes

P :	The number in parameter 'result' must be output after txt1 and txt2:
	outputres = 0 : no
	outputres = 1 : yes

R1 :	These three places must be filled in as 4, 5, or 6.
R2 :	The value of parameter 'textmode' gives the meaning of the contents of parameter 'text' : textmode = 2 : byte 1-80 of 'text' is a replace-command, i.e. a call of another supervisor program. The command must be terminated by a null-character. textmode = 3 : byte 1-80 of 'text' is a text (terminated by a null-character) to be output on the supervisor key station display instead of standard text(s). Byte 81-160 of 'text' is a replace-command (terminated by a null-character).

APPENDIX E: STANDARD TEXT NUMBERS

In the Data Entry system the following texts with the corresponding numbers are available to be used in the code procedure 'return':

Textnumber	Text	Textnumber	Text
1	stop	26	length
2	printer	27	next
3	break	28	punch
4	syntax	29	transmit
5	batch	30	erase
6	state	31	date
7	error	32	tape ser
8	magtape	33	number
9	load err	34	file gen
10	** supv	35	informa
11	not name	36	real seq
12	ok	37	block
13	cf list	38	factor
14	cf list;	39	load
15	disc	40	table
16	file	41	full
17	ident	42	printout
18	unknown	43	in use
19	exist	44	format
20	no room	45	job
21	name	46	subprogr
22	chars	47	disctabl
23	copied	48	reader
24	not	49	library
25	record		

APPENDIX F: PROGRAM EXAMPLE VOLUME

This program example shows the communication between the program and the operator by using the code procedures 'get command' and 'return'.

The program returns only with 'action' = continue, i.e. no check of the new input shall be made, and therefore the code procedure 'get parameter' is not used. This is done because the operator may key letters as well as digits after the text 'magtape number =' has appeared on the display screen, so the new input may not fullfil the syntax rules.

In the giveup-procedure for magtape, procedure 'mterror', is zmode set to zero before the jump to the call of CLOSE, which will release the driver process if an error on the magtape occurs.

AUTHOR: NNN

EDITED: YY.MM.DD

PROGRAM RC36-XXXXX.YY

VOLUME - BOSS-LABEL

KEYWORDS: MUSTL,MTA,DATA ENTRY - REL 2,LISTING

ABSTRACT: THIS PROGRAM INITIALIZES A MAGTAPE BY WRITTING A VOL1-LABEL AS THE FIRST BLOCK FOLLOWED BY TWO TAPEMARKS. THE LABEL IS A BOSS-LABEL. THIS PROGRAM IS A DATA ENTRY SUPERVISOR PROGRAM.

RCSL: XX-NNYYYY: ASCII SOURCE TAPE.
RCSL: XX-NNYYYY: REL. BIN. TAPE.
!

```
TITLE:          VOLUME- BOSS LABEL

ABSTRACT:       THIS PROGRAM INITTIALIZES A MAGTAPE BY WRITTING
                A VOL1 LABEL AS THE FIRST BLOCK FOLLOWED BY TWO
                TAPEMARKS. THE LABEL IS A BOSS-LABEL.
                THIS PROGRAM IS A DATA ENTRY SUPERVISOR PROGRAM.

SIZE:           1446 BYTES. INCLUDING ONE 84 BYTES MT-BUFFER.

DATE:

CALL:           VOLUME

THE PROGRAM ASKS FOR:
  MAGTAPE NUMBER   STRING OF 6 CHARACTERS.

OUTPUT MESSAGES:
  SYNTAX        SYNTAX ERROR IN CALL-LINE
  MAGTAPE ERROR <CODE>
                CONSULT THE APPENDIX TO THE RC3600 DATA ENTRY
                SYSTEM USER'S MANUAL.
  OK            PROGRAM EXECUTION IS TERMINATED SUCCESFULLY.

SPECIAL REQUIREMENTS:
                CMMD: (R0001: RCSL: 43-RI0398).
                GTPM: (R0003: RCSL: 43-RI0654).
                RETUR: (R0004: RCSL: 43-RI0528).

!
```

```
CONST
VOL1=         '<86><79><76><49>',
SP=           '<32>',
LABEL=        '                                    <48><48><48><48><48><48>
                                                 <94><94><94><94>',
AGNO=         'MAGTAPE NUMBER=<0>';


VAR
COMLINE:      STRING(112);       ! COMMAND LINE !
PRETUR:       INTEGER;           ! PARAM ADDR OF MESS-BUF !
PTEXT:        STRING(6);         ! PARAM TEXT !
PVALUE:       INTEGER;           ! PARAM VALUE !
PKIND:        INTEGER;           ! PARAM KIND !
DISPLAYTEXT: STRING(80);         ! RETURN SPECIAL TEXT!
PSEP:         INTEGER;           ! PARAM SEPERATOR !
RTEXT1:       INTEGER;           ! RETURN TEXT1 !
RTEXT2:       INTEGER;           ! RETURN TEXT2 !
RSPEC:        INTEGER;           ! RETURN SPECIAL !
RCON:         INTEGER;           ! RETURN CONTINUE !
RESULT:       INTEGER;           ! RETURN RESULT !
TEXTMODE:      INTEGER;          ! RETURN PARAMETER MODE!
ACTIONS:       INTEGER;          ! RETURN ACTIONS!
ELPST:         STRING(1);        ! HELP VARIABLES !
IDENT:         STRING(6);
X:             INTEGER;
I:             INTEGER;
OWNER:         STRING(6);
EXPLENGTH:     INTEGER;
```

```
MTO:         FILE                 ! MAGTAPE DESCRIPTION !
             'MTO',
             14,                  ! REPEAT,POSITION,BLOCKED !
             1,                   ! BUFFERS !
             84,                  ! SHARESIZE !
             UB;                  ! UNDEF. BLOCKED !
             GIVEUP MTERROR,      ! ERROR PROCEDURE !
             8'173637             ! GIVEUP MASK !
             OF RECORD            ! RECORD STRUCTURE !
               TOTAL:    STRING(84);
               LIDENT:   STRING(4) FROM 1;
               FIDENT:   STRING(6) FROM 5
             END;

PROCEDURE CMMD(VAR COMLINE: STRING(112);    ! GET COMMAND !
                 VAR PRETUR: INTEGER);
CODEBODY;

PROCEDURE GTPM(VAR COMLINE: STRING(112);    ! GET PARAMETER !
                 VAR PTEXT: STRING(6);
                 VAR PVALUE: INTEGER;
                 VAR PKIND: INTEGER;
                 VAR PSEP: INTEGER);
CODEBODY;

PROCEDURE RETUR(VAR PRETUR: INTEGER;        ! RETURN !
                  VAR RESULT: INTEGER;
                  VAR ACTIONS: INTEGER;
                  VAR TEXTMODE:INTEGER;
                  VAR TEXT:    STRING(30));
CODEBODY;


PROCEDURE MTERROR;
BEGIN
    RTEXT1 := 8;                            ! MAGTAPE !
    RTEXT2 := 7;                            ! ERROR !
    RSPEC := 1;                             ! OUTPUT CONTENTS OF RESULT !
    RCON := 0;                              ! DO NOT CONTINUE !
    RESULT := MTO.ZO;
    MTO.ZMODE := 0;
    GOTO 200;
END;

PROCEDURE SYNTAX;
BEGIN
    RTEXT1 := 4;                            ! SYNTAX !
    RTEXT2 := 0;                            ! NO TEXT 2 !
    RSPEC := 0;                             ! NO SPECIAL ACTION !
    RCON := 0;                              ! DO NOT CONTINUE !
    RESULT := 0;
    GOTO 300;
END;
```

```
BEGIN
    CMMD(COMLINE,PRETUR);
    GTPM(COMLINE,PTEXT,PVALUE,PKIND,PSEP);
    IF PSEP <> 2 THEN SYNTAX;
    DISPLAYTEXT := MAGNO;  EXPLENGTH:= 6;
 0: TEXTMODE := 1;                          ! OUTPUT TEXT IN DISPLAYTEXT !
    ACTIONS := 1;                           ! ACTION := CONTINUE !
    RETUR(PRETUR,RESULT,ACTIONS,TEXTMODE,DISPLAYTEXT); ! RETURN !
    CMMD(COMLINE,PRETUR);                   ! GET NEW INPUT !
            X := 0;
            WHILE X < EXPLENGTH DO BEGIN
                MOVE(COMLINE,X,HELPST,0,1);
                IF BYTE HELPST = 0 THEN BEGIN ! CHANGE NULL-CHARS TO SPACES !
                    REPEAT MOVE(SP,0,IDENT,X,1);
                    X := X + 1 UNTIL X = EXPLENGTH;
                    GOTO 20;
                END;
                MOVE(HELPST,0,IDENT,X,1);
                X := X + 1;
30:         END;
20: MOVE(IDENT,0,OWNER,0,6);
    OPEN(MT0,3);
    SETPOSITION(MT0,1,1);
    PUTREC(MT0,84);
    INSERT(64,MT0↑,0);  INSERT(64,MT0↑,1);
    MOVE(MT0↑,0,MT0↑,2,82);
    MT0↑.LIDENT := VOL1;
    MT0↑.FIDENT := OWNER;

    MOVE(LABEL,0,MT0↑,10,74);
    OUTBLOCK(MT0);
    RTEXT1 := 12;                           ! TEXT1 := OK !
    RTEXT2 := 0;                            ! NO TEXT2 !
    RSPEC := 0;                             ! NO SPECIAL ACTION !
    RCON := 0;                              ! DO NOT CONTINUE !
    RESULT := 0;
200:
    CLOSE(MT0,1);
300:
    ! PACK TEXTS IN ACTIONS !
    ACTIONS := RTEXT1 SHIFT 10 + RTEXT2 SHIFT 4 + RSPEC SHIFT 1 + RCON;
    TEXTMODE := 0;
    RETUR(PRETUR,RESULT,ACTIONS,TEXTMODE,DISPLAYTEXT);
 ND;
```

APPENDIX G: PROGRAM EXAMPLE DUMP

This program example shows how to use all the standard code procedures except procedure allaccess (this is used in the same way as procedure access), and it shows how to make a printout by 'outtext'.

The main program starts by getting the command line into 'comline'. Then it initializes all the error-counters to 4. When a zone is opened the corresponding error-counter is set to zero and when the zone is closed, it is set to 4 again.

The parameters in 'comline' is received by calling the code procedure 'get parameter' (gtpm). First call gives the name of the printout file and it is saved in 'lname', which is used in the call of code procedure 'connect file' (connec).

Most of the texts written with 'outtext' are constant texts specified in the constant section of the program. They are terminated by invisible null-characters and 'outtext' will therefore stop on these null-characters. But if a constant string is moved to another string, this null-character is lost and must be inserted as the last character. When writing the constant line 'headline' the name from the command line is inserted and to be sure that eventual null-characters in the name will not stop 'outtext' the procedure 'outname' is called to changed the null-characters to spaces.

When all of the batches in a job must be dumped, the job is opened and the batch names are found one by one by calling the code procedure 'get next item' (getnex). If the zone has not been destroyed between two calls, the parameter 'res' (corresponding to 'status') must be 1 before the call (i.e. if the batch does not exist as a disc-file or if the batch is used by another key station).

The procedure 'dumpbatch' opens the batch and checks the batch head. Then it reads the data in the batch by calling the code procedure 'access' (acces), and when a new block is read, the program must wait some timer-periods by calling the code procedure 'delay'.

Before returning to the Supervisor by calling the code procedure 'return' (retur) the program must release all drivers and area processes, but only if the error-counter is smaller than 4. This will avoid a program loop if an error has occurred. Then the parameter 'rtext1' (corresponding to 'action') is packed and the call of 'return' with continue = 0 will return to the Supervisor.

!

RCSL: XX-NNYYYY

AUTHOR: NNN

EDITED: YY.MM.DD

PROGRAM RC36-XXXXX.YY

DUMP - RELEASE 2.

KEYWORDS: MUSIL,CONVERSION,DPO,MTA,PRINTOUT,DATA ENTRY,LISTING

BSTRACT: THIS PROGRAM HANDLES DATA BATCHES OR JOBS OF DATA BATCHES FROM DISC WITH A MAXIMUM RECORDSIZE OF 79 BYTES WITH ASCII CODE DATA.
OUTPUT ON A LABELLED TAPE WITH A BLOCKSIZE OF 720 BYTES, IN ASCII CODE, EACH BLOCK CONSISTING OF 9 RECORDS OF 80 BYTES IN ASCII CODE.
THE LAST BYTE IS FILLED UP WITH THE CHARACTER NL.
IF A RECORD IS SMALLER THAN 79 BYTES IT IS FILLED UP WITH SPACES.
IF A RECORD IS GREATER THAN 79 BYTES IT IS CUT OFF AND THE TEXT 'RECORD <NO> TOO LONG' IS WRITTEN IN THE PRINTOUT.
THE LABEL IS A STANDARD BCSS-LABEL.
INVALID RECORDS ARE NOT DUMPED.
NUMBER OF DUMPED RECORDS IS WRITTEN IN THE PRINTOUT.
THIS PROGRAM IS A DATA ENTRY SUPERVISOR PROGRAM.

RCSL: XX-NNYYYY: ASCII SOURCE TAPE
RCSL: XX-NNYYYY: REL.BIN TAPE
!

```
TITLE:                 DUMP - RELEASE 2.

ABSTRACT:              THIS PROGRAM HANDLES DATA BATCHES OR JOBS OF DATA BATCHES
                       FROM DISC WITH A MAXIMUM RECORDSIZE OF 79 BYTES WITH ASCII
                       CODE DATA.
                       OUTPUT ON  LABELLED TAPE WITH A  BLOCKSIZE OF 720 BYTES,
                       IN ASCII CODE, EACH BLOCK CONSISTING OF 9
                       RECORDS OF 80 BYTES IN ASCII CODE.
                       IF A RECORD IS SMALLER THAN 79 BYTES IT IS FILLED UP
                       WITH SPACES.
                       IF A RECORD IS GREATER THAN 79 BYTES IT IS CUT OFF AND
                       THE TEXT 'RECORD <NO> TOO LONG' IS WRITTEN IN THE PRINTOUT.
                       THE LABEL IS STANDARD BOSS-LABEL.
                       INVALID RECORDS ARE NOT DUMPED.
                       NUMBER OF DUMPED RECORDS IS WRITTEN IN THE PRINTOUT.
                       THIS PROGRAM IS A DATA ENTRY SUPERVISOR PROGRAM.

SIZE:                  7860 BYTES. INCLUDING ONE 512 BYTES INPUT BUFFER,
                       ONE 720 BYTES OUTPUT BUFFER AND ONE PRINTOUT-BUFFER.

DATE:

CALL:                  DUMP  BATCH/JOB  <BATCHNAME>/<JOBNAME>  NEW/OLD
                       BATCH/JOB:  INDICATES IF A BATCH OR A JOB IS TO BE DUMPED.
                       <BATCHNAME>/<JOBNAME>: THE NAME OF THE BATCH OR THE JOB
                            THAT IS TO BE DUMPED.
                       NEW/OLD: NEW STARTS DUMPING AFTER THE BOSS-LABEL.
                            OLD STARTS DUMPING AFTER THE LAST DATABLOCK.
                      NEW/OLD CAN BEE FOLLOWED BY DUMPOK AND/OR RELEASE:
                      DUMPOK: THE BATCH/JOB HAS BEEN DUMPED ONCE BEFORE,
                              AND MUST BE DUMPED ONCE MORE.
                      RELEASE: THE BATCH/JOB MUST BE REKEYED AND NOT RE-
                               KEYED YET.


OUTPUT MESSAGES:
  NOT NAME             NO BATCH WITH THE SPECIFIED NAME EXISTS.
  NOT BATCH            THE SPECIFIED BATCHNAME IS NOT A BATCH WITHIN THE SYSTEM.
  SYNTAX               SYNTAX ERROR IN THE CALL LINE.
  MAGTAPE UNKNOWN
                       THE MAGTAPE IS NOT INITIALIZED BY A VOL1-LABEL.
  CF LIST              SOME ERRORMESSAGES ARE WRITTEN IN THE PRINTOUT.
                       CONFER THE  LOG.
  BATCH IN USE         THE BATCH IS USED BY ANOTHER KEYSTATION.
  JOB IN USE           THE JOB IS USED BY ANOTHER KEYSTATION.
  LIBRARY IN USE THE JOBLIBRARY IS USED BY ANOTHER KEYSTATION.
  NOT JOB              TH SPECIFIED NAME IS NOT A NAME OF A JOB.
  OK                   PROGRAM EXECUTION IS TERMINATED SUCCESFULLY.
  MAGTAPE ERROR <CODE>  CONSULT THE RC3600 DATA ENTRY USER'S
                             GUIDE, PART 2, APPENDIX 3.
  DISC ERROR <CODE> CONSULT THE RC3600 DATA ENTRY USER'S
                        GUIDE, PART 2, APPENDIX 2.
  PRINTOUT ERROR <CODE> CONSULT THE RC3600 DATA ENTRY USER'S
                            GUIDE, PART 2, SECTION 8.2.

SPECIAL REQUIREMENTS:
                       CMMD ( R0001: RCSL: 43-RI0398 ).
                       GTPM ( R0003: RCSL: 43-RI0654 ).
                       RETUR ( R0004: RCSL: 43-RI0528 ).
                       ACCES ( R0013: RCSL: 43-RI0796).
                       CONNEC (P0086: RCSL: 43-GL3275).
                       FITEM (R0017:  RCSL: 43-RI01034).
                       GETNEX (R0018: RCSL: 43-RI0931).
                       DELAY (R0021: RCSL: 43-RI0995).
!
```

```
           ! 9 RECORD PER BLOCK !
CONST
NEW=       'NEW',
OLD=       'OLD',
ECLGT=     79,                        !CONSTANT RECORD LENGTH!


VOL1=       '<86><79><76><49>',
JBLIB=     'JBLIB<0>',                ! NAME OF JOB LIBRARY !
FF=        '<12>',
JOB=       'JOB',
HBATCH=    'BATCH',
HJOB=      'JOB  ',
TXNLSP=    '<13><10>  ',
TXSPACES=  '<13><10> NO OF DUMPED RECORDS: ',
SPACES=    '      ',
EM=        '<25>',                    ! END MEDIUM CHARACTER !
NULL=      '<0>',
NOTBATCH=  ':  NOT BATCH<13><10>',
TXEX=      ':  DOES NOT EXIST<13><10>',
STATERR=   ':  NOT DUMPED, STATE ERROR<13><10>',
BUSED=     ':  NOT DUMPED, IN USE<13><10>',
NOREK=     'RELEA',
MPOK=      'DUMPO',


HEADLINE=  '<13><10>* * * * * * * * * * *  DUMP OF
* * * * * * * * * * *',
TEXTLINE=  '<13><10>BATCHNAME  ',
TEXTLINE1= '<13><10>FORMAT     ',
TXTOTAL=   '<13><10>THE TOTAL NUMBER OF DUMPED RECORDS:          ',
TXEND=     '<13><10>* * * * * * * * * * *      END OF DUMP
* * * * * * * * * * *<13><10>',
TXLONG=    '<13><10>RECORD NUMBER       TOO LONG',
TXINVAL=   '<13><10>RECORD NUMBER       INVALID, NOT DUMPED',
NL=        '<10>',
SP=        '<32>';
```

```
VAR
COMLINE:      STRING(112);                 !  COMMAND LINE
PRETUR:       INTEGER;                     !  PARAMETER ADDR OF MESSAGE BUF       !
PTEXT:        STRING(6);                   !  PARAMETER TEXT                      !
PVALUE:       INTEGER;                     !  PARAMETER VALUE
PKIND:        INTEGER;                     !  PARAMETER KIND                      !
PSEP:         INTEGER;                     !  PARAMETER SEPERATOR                 !
PERIODS:      INTEGER;                     !  RETURN PARAMETER                    !

BLOCK:        INTEGER;                     !  NUMBERS OF BLOCKS ON OLD TAPE       !
FIELD:        STRING(80);                  !  FIELD IN RECORD                     !
SUBNAME:      STRING(2);                   ! SUBNAME OF FORMAT OF RECORD          !

X:            INTEGER;                     !  NUMBER OF CHAR IN BUFFER            !
Y:            INTEGER;                     !  NUMBER OF CHAR IN FIELD             !
V:            INTEGER;
Z:            INTEGER;
I:            INTEGER;
DBLOCK:       INTEGER;                     ! BLOCK NUMBER IN BATCH                !


EOF:          INTEGER;

COM:          STRING(5);                   !  COMMAND JOB OR BATCH                !
JOBNAME:      STRING(6);                   !  JOBNAME OR BATCHNAME                !
EWORD:        INTEGER;                     !  NB!! JOBNAME, EWORD AND RES
RES:          INTEGER;                     !  MUST BE CONSECUTIVE                 !
CTYPE:        INTEGER;                     ! COMMAND TYPE IN CODE PROC. FITEM     !
LENGTH:       INTEGER;                     ! LENGTH OF ITEM IN JOB OR LIBRARY     !
STATUS:       INTEGER;                     !  BATCH STATUS
SUM:          INTEGER;                     !  TOTAL DUMPED RECORDS                !
REC:          STRING(80);                  !  WORKING AREA FOR CURR.RECORD        !
DUMPNO:       INTEGER;                     !  NUMBER OF DUMPED RECORDS            !
ITEM1:        STRING(5);
ITEM2:        STRING(6);
FLAG:         INTEGER;
RELEASE:      INTEGER;
NOHELP:       STRING(2);
NO:           INTEGER;                     !  NUMBER OF CHAR IN REC               !
RECNO:        INTEGER;                     !  CURR. RECORD NUMBER                 !
BINNO:        INTEGER;                     !  WORKING AREA
NAME:         STRING(6);                   !  BATCH NAME !
DECNO:        STRING(5);                   !  WORKING LOCATION                    !
ERRLPT:       INTEGER;                     !  NUMBER OF ERRORS ON LPT             !
ERRBAT:       INTEGER;                     !  NUMBER OF ERRORS ON BATCH           !
ERRJAT:       INTEGER;                     !  NUMBER OF ERRORS ON JOB             !
ERRMTO:       INTEGER;                     !  NUMBER OF ERRORS ON MTO             !

RTEXT1:       INTEGER;                     !  RETURN PARAMETER TEXT1              !
RTEXT2:       INTEGER;                     !  RETURN PARAMETER TEXT2              !
RSPEC:        INTEGER;                     !  RETURN PARAMETER SPECIAL            !
RCON:         INTEGER;                     !  RETURN PARAMETER CONTINUE           !
RESULT:       INTEGER;                     !  RETURN PARAMETER RESULT             !
TEXTMODE:      INTEGER;                    !  RETURN PARAMETER MODE               !
LNAME:        STRING(6);                   !  NAME OF PRINTOUT-FILE               !
```

```
LPT:        FILE                        !  PRINTER FILE DESCRIPTION              !
            'LPT',                      !  NAME OF PRINTER DRIVER                !
            62,                         !  KIND = CHARACTER ORIENTED             !
            1,                          !  BUFFERS      !
            512,                        !  SHARESIZE    !
            U;                          !  FORMAT = UNDEFINED                    !

            GIVEUP
            LPTERROR,                   !  LPT ERROR PROCEDURE                  !
            2'1111111111111111

            OF STRING(512);             !  RECORD STRUCTURE                      !


BATCH:      FILE                        !  BATCH ERROR DESCRIPTION               !
            'BATCH',                    !  NAME OF BATCH                         !
            60,                         !  KIND = POSITIONABLE, REPEATABLE       !
            1,                          !  BUFFERS      !
            512,                        !  SHARESIZE    !
            U;                          !  FORMAT = UNDEFINED                    !

            GIVEUP
            BATERROR,                   !  BATCH ERROR PROCEDURE                 !
            2'1111111111111111

            OF RECORD                   !  RECORD STRUCTURE                      !
                 JNAME:  STRING(6) FROM 1;
                 FORM:   STRING(6) FROM  7;
                 BNAME:  STRING(3) FROM 13;
                 STAT:  STRING(2) FROM 19
               END;


MTO:        FILE
            'MTO',
            14,                         !  KIND = REPEATABLE, POSITIONABLE,      !
                                        !         BLOCKED                        !
            1,                          !  BUFFERS      !
            720,                        !  SHARESIZE    !
            FB;                          !  FORMAT = FIXED BLOCKED


            GIVEUP
            MTERROR,                    !  MT ERROR PROCEDURE                    !
            2'1111011110011110

            OF STRING(80);            !  RECORD STRUCTURE
```

```
PROCEDURE CMND (VAR COMLINE:  STRING(112);  !  GET COMMAND                    !
               VAR PRETUR:   INTEGER     );
CODEBODY;

PROCEDURE GTPM (VAR COMLINE:  STRING(112);  !  GET PARAMETER                  !
               VAR PTEXT:    STRING(5)   ;
               VAR PVALUE:   INTEGER     ;
               VAR PKIND:    INTEGER     ;
               VAR PSEP:     INTEGER     );
CODEBODY;

PROCEDURE RETUR(VAR PRETUR:   INTEGER     ;  !  RETURN                         !
               VAR RESULT:   INTEGER     ;
               VAR RTEXT1:   INTEGER     ;
               VAR TEXTMODE:  INTEGER     ;
               VAR COMLINE:  STRING(112));
CODEBODY;

PROCEDURE ACCES(FILE BATCH;                 ! GET PART OF RECORD !
               VAR FIELD:    STRING(80);
               VAR Y:        INTEGER     ;
               VAR STATUS :  INTEGER     ;
               VAR  SUBNAME: STRING(2)) ;
CODEBODY;

PROCEDURE CONNEC(FILE LPT                ;   ! CONNECT PRINTOUT !
               VAR X:       INTEGER     ;
               VAR LNAME:   STRING(6))  ;
CODEBODY;

PROCEDURE FITEM(FILE BATCH               ;   ! FIND ITEM !
               CONST CTYPE: INTEGER     ;
               CONST LENGTH:INTEGER     ;
               VAR   NAME:  STRING(6))  ;
CODEBODY;


PROCEDURE GETNEX(FILE BATCH              ;   ! GET NEXT ITEM !
               CONST LENGTH:INTEGER     ;
               VAR   NAME:  STRING(6)   ;
               VAR   EWORD: INTEGER     ;
               VAR   RES:   INTEGER   ) ;
CODEBODY;
PROCEDURE DELAY(VAR PERIODS: INTEGER;       ! DELAYS THE CALLING PROGRAM !
               VAR PRETUR:  INTEGER);
CODEBODY;
```

```
! PROCEDURE REPOS REPOSITIONS THE MAGTAPE JUST AFTER THE
  LAST BATCH, WHICH HAS BEEN FINISHED AND IS CALLED FROM THE
  THREE GIVEUP-PROCEDURES: LPTERROR, BATERROR AND MTERROR !

PROCEDURE REPOS;
BEGIN
    SETPOSITION(MT0,2,BLOCK);
    IF RTEXT1 = 42 THEN GOTO 400;           ! IF THE PROCEDURE IS CALLED
                                              FROM PROCEDURE LPTERROR !
    DUMPNO := 0;
    OUTTEXT(LPT,NL);
    OUTTEXT(LPT,TXSPACES);
    BINDEC(DUMPNO,DECNO);
    OUTTEXT(LPT,DECNO);       OUTTEXT(LPT,NL);
    IF COM = JOB THEN BEGIN
        BINDEC(SUM,DECNO);
        OUTTEXT(LPT,TXTOTAL);
        OUTTEXT(LPT,DECNO);
        OUTTEXT(LPT,NL);
    END;
    OUTTEXT(LPT,TXEND);
400:
    GOTO 200;
END;

! PROCEDURE LPTERROR IS THE GIVEUP-PROCEDURE FOR THE
  PRINTOUT-FILE !

PROCEDURE LPTERROR;
BEGIN
    RTEXT1 := 42;                           !  PRINTOUT    !
    RTEXT2 := 7;                            !  ERROR      !
    RSPEC  := 1;                            ! OUTPUT CONTENTS OF RESULT !
    RCON   := 0;                            ! DO NOT CONTINUE !
    RESULT := LPT.Z0;
    ERRLPT := ERRLPT + 1;
    REPOS;
!   DO NOT COPY EVENTUAL PRINTOUT TO DISPLAY SCREEN OR
    HARDCOPY DEVICE !
    FLAG:= 0;
    GOTO 200;
END;

! PROCEDURE OUTNAME CHANGES THE EVENTUAL NULL-CHARACTERS
  IN A NAME IN "ITEM1" TO SPACES AND INSERTS A NULL-CHA-
  RACTER AS THE LAST CHARACTER AND PLACES THE RESULT IN "ITEM2".
  THEN A CALL OF "OUTTEXT" WILL WRITE 5 CHARACTERS NO MATTER
  HOW MANY CHARACTERS THERE WERE IN "ITEM1". !

PROCEDURE OUTNAME;
BEGIN
   V:= 0;
   WHILE V < 5 DO
      BEGIN
        MOVE(ITEM1,V,DECNO,0,1);
        IF DECNO = NULL THEN MOVE(SP,0,DECNO,0,1);
        MOVE(DECNO,0,ITEM2,V,1);
        V := V + 1;
      END;
   MOVE(NULL,0,ITEM2,5,1);
END;
```

```
! PROCEDURE CHECKSTAT CHECKS THE STATUS OF THE BATCH
  BEFORE DUMP. !
PROCEDURE CHECKSTAT;
BEGIN
   IF STATUS AND 8'174000 = 8'100000 THEN GOTO 60;     !CLOSED !
   IF STATUS = 0 THEN GOTO 63;                         !EMTY   !
   GOTO 62;
60:IF RELEASE = 3 THEN GOTO 63;                        !NO FURTHER CHECK!
   IF RELEASE = 2 THEN GOTO 61;
   IF STATUS AND 8'3000 = 8'2000 THEN GOTO 62;         !MUST BE REKEYED!
                                                       !AND REKEYED    !
   IF STATUS AND 2 = 2 THEN                            !VALID REQUIRED!
   IF STATUS AND 8'10 <> 0 THEN GOTO 62;               !AND NOT INVALID!
61:IF RELEASE = 1 THEN GOTO 63;
   IF STATUS AND 8'40 = 8'40 THEN GOTO 62;             !DUMPED!
   GOTO 63;
62:OUTTEXT(LPT,STATERR);                               !STATE ERROR!
   STATUS := 0;                                        !MAY NOT BE DUMPED!
   RTEXT1 := 13;                                        ! TEXT1 := CF LIST !
   GOTO 64;
63:STATUS := 1;                                         ! STATUS OK. !
64:END;

! PROCEDURE DUMPBIT SETS THE BIT "HAS BEEN DUMPED" IN THE
  BATCH STATUS WORD IN THE BATCH HEAD. !

PROCEDURE DUMPBIT;
BEGIN
   SETPOSITION(BATCH,0,0);
   GETREC(BATCH,NO);
   STATUS := WORD BATCH↑.STAT;
   IF STATUS AND 8'40 = 0 THEN BEGIN
     NO := STATUS + 8'40;
     CLOSE(BATCH,1);
     OPEN(BATCH,3);
     SETPOSITION(BATCH,0,0);
     BATCH.ZFIRST := BATCH.ZTOP;
     MOVE(NOHELP,2,BATCH↑,18,2);
     BATCH.ZREM := 0;
     OUTBLOCK(BATCH);
   END;
END;

! PROCEDURE BATCHSTATE IS CALLED WHEN A BATCH OR JOB FILE
  DOES NOT EXIST, I.E. FROM PROCEDURE BATERROR WHEN STATUS=8'40000. !

PROCEDURE BATCHSTATE;
BEGIN
    IF COM = HBATCH THEN BEGIN
       RTEXT1 := 5;                             !BATCH!
55:    RTEXT2 := 43;                            ! IN USE !
       RSPEC := 0;                              ! NO SPECIAL ACTION !
       RCON := 0;                               ! DO NOT CONTINUE !
       RESULT := 0;
       FLAG := 0;
       GOTO 200;
       END;
    IF BATCH.ZNAME = JOBNAME THEN BEGIN
       RTEXT1 := 45;                            !JOB!
       GOTO 55;
       END;
    OUTTEXT(LPT,BUSED);
    RTEXT1 := 13;
    RES := 2;
    GOTO 70;
END;
```

```
! PROCEDURE BATERROR IS THE GIVEUP-PROCEDURE FOR THE DISC. !

PROCEDURE BATERROR;
BEGIN
    RSPEC  := 1;
    RCON   := 0;
    RESULT := BATCH.Z0;
    IF BATCH.Z0 AND 8'1000 <> 0 THEN BATCHSTATE;  ! FILE IN USE !
    IF BATCH.Z0 AND 8'040000 <> 0 THEN            ! FILE DOES NOT EXIST !
    BEGIN
     IF FLAG = 0 THEN BEGIN
        RTEXT1 := 11;                        !  NOT NAME !
        RTEXT2 := 0;
        RSPEC := 0;
        GOTO 10;
        END;
    OUTTEXT(LPT,TXEX);
    RTEXT1:= 13;
    RES:= 2;
    GOTO 70;
     END;
    RTEXT1 := 15;                            !  DISC      !
    RTEXT2 := 7;                             !  ERROR     !
    REPOS;                                   ! REPOSITION MAGTAPE !
10: IF BATCH.ZNAME <>JOBNAME THEN ERRBAT:= ERRBAT+1;
    IF BATCH.ZNAME = JOBNAME THEN ERRJAT:= ERRJAT+1;
    GOTO 200;
END;

! PROCEDURE MTERROR IS THE GIVEUP-PROCEDURE FOR THE MAGTAPE !

PROCEDURE MTERROR;
BEGIN
   IF MT0.Z0 AND  8'10  <> 0 THEN GOTO 510; ! POSITION ERROR !
   IF MT0.Z0 AND 8'400 = 0 THEN BEGIN       ! IF NOT END OF FILE !
          RTEXT1 := 8;                       !  MAGTAPE  !
          RTEXT2 := 7;                       !  ERROR    !
          RSPEC  := 1;                       ! OUTPUT CONTENTS OF RESULT !
          RCON   := 0;                       ! DO NOT CONTINUE !
          RESULT := MT0.Z0;
          ERRMT0 := ERRMT0 + 1;
          ! IF NOISE RECORD, BLOCK LENGTH, DATA CHANNEL, PARITY, EOT !
          ! THEN REPOSITION MAGTAPE !
          IF MT0.Z0 AND 8'10360 <> 0 THEN REPOS;
          GOTO 200;
   END;
   IF MT0.Z0 AND 8'400 <> 0 THEN EOF := 1; ! END OF FILE !
510:
END;

! PROCEDURE ERRHEAD IS CALLED IF THE MAGTAPE IS NOT
  HEADED IN THE RIGHT WAY .!

PROCEDURE ERRHEAD;
BEGIN
    RTEXT1 := 8;           ! MAGTAPE !
    RTEXT2 := 18;          ! UNKNOWN !
    RSPEC := 0;            ! NO SPECIAL ACTION !
    RCON := 0;             ! DO NOT CONTINUE !
    RESULT := 0;
    GOTO 200;
END;
```

```
! PROCEDURE HEADER CHECKS THAT THE MAGTAPE IS HEADED IN
  THE RIGHT WAY. !

PROCEDURE HEADER;
BEGIN
    OPEN(MT0,5);
    ERRMT0 := 0;
    Y := 84;
    SETPOSITION(MT0,1,1);
    GETREC(MT0,Y);
    IF MT0↑ <> VOL1 THEN ERRHEAD;
    INBLOCK(MT0);
    IF EOF <> 1 THEN ERRHEAD;
    IF ITEM2 = OLD THEN GOTO 1000;
    EOF := 0;
    INBLOCK(MT0);
    IF EOF <> 1 THEN ERRHEAD;
    CLOSE(MT0,0);
    OPEN(MT0,3);
    SETPOSITION(MT0,2,1);
    BLOCK := 0;
1000: END;

! PROCEDURE SYNTAX IS CALLED IF THE CALL OF THE
  PROGRAM HAS NOT THE RIGHT PARAMETERS. !

PROCEDURE SYNTAX;                                  ! PARAMETER ERROR !
BEGIN
    RTEXT1 := 4;                                   !  SYNTAX    !
    RTEXT2 := 0;                                   ! NO TEXT2 !
    RSPEC  := 0;                                   ! NO SPECIAL ACTION !
    RCON   := 0;                                   ! DO NOT CONTINUE !
    RESULT := 0;
    GOTO 300;
END;

! PROCEDURE NOTNAME IS CALLED WHEN A BATCH OR A JOB
  TO BE DUMPED IS FOUND NOT TO BE A BATCH OR A JOB. !

PROCEDURE NOTNAME;
BEGIN
    IF COM = JOB THEN  IF BATCH.ZNAME = NAME THEN BEGIN
    OUTTEXT(LPT,NOTBATCH);
    CLOSE(BATCH,1);  ERRBAT := 4;
    GOTO 70;
    END;
    RTEXT1 := 24;                                  !  NOT       !
    IF COM = JOB THEN RTEXT2 := 45;                !JOB !
    IF COM = HBATCH THEN RTEXT1 := 5;              !  BATCH !
    RSPEC  := 0;                                   ! NO SPECIAL ACTION !
    RCON   := 0;                                   ! DO NOT CONTINUE !
    RESULT := 0;
    GOTO 200;
END;
```

```
! PROCEDURE WRITEREC WRITES ONE RECORD ON THE MAGTAPE. !

PROCEDURE WRITEREC;
BEGIN
    IF NO <> 0 THEN
      BEGIN
        IF NO > RECLGT THEN BEGIN
             BINDEC(RECNO,DECNO);
             MOVE(DECNO,0,TXLONG,16,5);
             OUTTEXT(LPT,TXLONG);
             RTEXT1 := 13;
          END;
        WHILE NO < RECLGT DO BEGIN
            MOVE(SP,0,REC,NO,1);
            NO := NO + 1;
            END;
        DUMPNO := DUMPNO + 1;
        PUTREC(MT0,RECLGT+1);
        MOVE(REC,0,MT0↑,0,RECLGT);
        INSERT(10,MT0↑,79);
      END;
    NO := 0;
END;
```

```
 ! PROCEDURE DUMPBATCH IS THE PROCEDURE TO READ THROUGH A
   BATCH AND DUMP ALL THE DATA IN THE BATCH. !

 PROCEDURE DUMPBATCH;
 BEGIN
    OUTTEXT(LPT,TEXTLINE);  OUTTEXT(LPT,SPACES);
    ITEM1 := NAME;  OUTNAME;  OUTTEXT(LPT,ITEM2);
    BATCH.ZNAME := NAME;
    OPEN(BATCH,1);  ERRBAT := 0;              !  OPEN DISC FOR READING       !
    DBLOCK := 0;     RECNO := 1;   NO := 0;   DUMPNO := 0;
    SETPOSITION(BATCH,0,0);
    GETREC(BATCH,X);                          !  GET BATCHHEAD               !
    IF BATCH.ZNAME <> BATCH↑.BNAME THEN NOTNAME; ! CHECK BATCH HEAD !
    IF BATCH↑.JNAME <> JOBNAME THEN NOTNAME;
    STATUS := WORD BATCH↑.STAT;
    CHECKSTAT;                                ! CHECK STATUS OF BATCH !
    IF STATUS = 0 THEN GOTO 30;               ! STATE ERROR !
     OUTTEXT(LPT,TEXTLINE1);
     OUTTEXT(LPT,SPACES);
    ITEM1 := BATCH↑.FORM;  OUTNAME;  OUTTEXT(LPT,ITEM2);
    SETPOSITION(BATCH,0,1);
    DBLOCK := 1;
 5: ! IF BATCH.ZBLOCK HAS BEEN CHANGED THEN LET THE
      PROGRAM WAIT SOME TIMER-PERIODS !
    IF BATCH.ZBLOCK <> DBLOCK THEN DELAY(PERIODS,PRETUR);
    DBLOCK := BATCH.ZBLOCK;
    ACCES(BATCH,FIELD,Y,STATUS,SUBNAME);    ! GET PART OF BATCH !
    IF Y >= 0 THEN BEGIN                ! A FIELD IS READ, MOVE FIELD TO REC !
       V:=Y;
       IF NO + V > 79 THEN BEGIN
         V:=79-NO;
         IF V < 0 THEN V:=0;
       END;
       IF NO + Y <= RECLGT THEN MOVE(FIELD,0,REC,NO,V);
       Z := 79-NO;
       IF NO < RECLGT THEN IF NO + Y > RECLGT THEN MOVE(FIELD,0,REC,NO,Z);
       NO := NO + Y;
       GOTO 10;
    END;
    IF Y = -1 THEN BEGIN                ! A RECORD HEAD IS READ !
       IF STATUS <> 0 THEN BEGIN        ! IF RECORD STATUS ERROR THEN DO
                                          NOT DUMP THE RECORD, ONLY READ !
          BINDEC(RECNO,DECNO);
          MOVE(DECNO,0,TXINVAL,16,5);
          OUTTEXT(LPT,TXINVAL);
          REPEAT ACCES(BATCH,FIELD,Y,STATUS,SUBNAME)
          UNTIL Y = -2;
          RECNO := RECNO + 1;
       END;
       GOTO 10;
    END;
    IF Y = -2 THEN BEGIN                ! A RECORD END IS READ !
       WRITEREC;                        ! WRITE RECORD ON MAGTAPE !
       RECNO := RECNO + 1;
       GOTO 10;
    END;
    IF Y = -3 THEN GOTO 20;             ! A BATCH END IS READ !
 10:
    GOTO 5;
```

```
20:
      WHILE MT0.ZREM <> 0 DO BEGIN   ! FILL UP THE LAST BLOCK ON
                                       THE MAGTAPE WITH NULLS !
      PUTREC(MT0,80);
      INSERT(0,MT0↑,0);
      INSERT(0,MT0↑,1);
      MOVE(MT0↑,0,MT0↑,2,78);
     END;
      OUTBLOCK(MT0);
      OUTTEXT(LPT,TXNLSP);
      OUTTEXT(LPT,TXSPACES);
   BINDEC(DUMPNO,DECNO);             ! WRITE IN PRINTOUT NUMBER OF
                                       DUMPED RECORDS !
   OUTTEXT(LPT,DECNO);  OUTTEXT(LPT,NL);  OUTTEXT(LPT,NL);
   BLOCK := MT0.ZBLOCK;
   DUMPBIT;                          ! SET THE BIT "HAS BEEN DUMPED"
                                       IN BATCH STATUS WORD !
30:
   CLOSE(BATCH,1);  ERRBAT := 4;
END;



! PROCEDURE CHECKKIND CHECKS THAT THE CALL LINE PARAMETER
  "BATCH/JOB" AND THE NAME CORRESPONDS, I.E. FOR EXAMPLE A JOB MUST
  STAND IN JOB LIBRARY(JBLIB). !

PROCEDURE CHECKKIND;
BEGIN
   IF COM = HBATCH THEN
      BEGIN
         BATCH.ZNAME := NAME;
         OPEN(BATCH,1);  ERRBAT := 0;
         SETPOSITION(BATCH,0,0);  GETREC(BATCH,X);
         IF BATCH.ZNAME <> BATCH↑.BNAME THEN NOTNAME;
         MOVE(BATCH↑,0,JOBNAME,0,6);
         CLOSE(BATCH,1);  ERRBAT := 4;
     END;
   BATCH.ZNAME := JBLIB;
   OPEN(BATCH,1);  ERRBAT := 0;
   SETPOSITION(BATCH,0,0);
   ! LENGTH OF JOB IN JBLIB = 3 WORDS.
     COMMAND TYPE := SEARCH !
   LENGTH := 3;  CTYPE := 0;
   FITEM(BATCH,CTYPE,LENGTH,JOBNAME);  ! FIND JOB IN JBLIB !
   IF RES = 1 THEN NOTNAME;            ! IF NOT FOUND THEN ERROR !
   CLOSE(BATCH,1);  ERRBAT := 4;
END;
```

```
                              !  DUMP MAIN PROGRAM START !
BEGIN
   CMMD(COMLINE,PRETUR);                          !  GET COMMAND                 !
   ERRJAT:=4;  ERRMTO:=4;  ERRBAT:=4;  ERRLPT:=4;
   GTPM(COMLINE,PTEXT,PVALUE,PKIND,PSEP);   !  GET PARAM.-PRINTOUT FILE NAME!
   LNAME:= PTEXT;                                ! SAVE PRINTOUT FILE NAME !
   FLAG:= 0;  RELEASE:= 0;
   GTPM(COMLINE,PTEXT,PVALUE,PKIND,PSEP);   !  GET PARAMETER - MUST BE      !
                                            !            JOB / BATCH        !
   IF PKIND <> 1 THEN SYNTAX;               !             ELSE SYNTAX       !
   IF PSEP = 2 THEN SYNTAX;
   COM := PTEXT;
   IF COM <> HBATCH THEN IF COM <> JOB THEN SYNTAX;
   GTPM(COMLINE,PTEXT,PVALUE,PKIND,PSEP);   !  GET PARAMETER - MUST BE      !
                                            !         JOB- OR BATCHNAME     !
   IF PKIND <> 1 THEN SYNTAX;
   IF COM = HBATCH THEN NAME := PTEXT;
   IF COM = JOB THEN JOBNAME := PTEXT;
   BATCH.ZNAME := PTEXT;
   OPEN(BATCH,1);  ERRBAT := 0;             ! CHECK IF NAME EXISTS !
   CLOSE(BATCH,1);  ERRBAT := 4;
   GTPM(COMLINE,PTEXT,PVALUE,PKIND,PSEP);   !  GET PARAMETER  OLD/NEW       !
   IF PKIND <> 1 THEN SYNTAX;
   ITEM2 := PTEXT;
   GTPM(COMLINE,PTEXT,PVALUE,PKIND,PSEP);   ! GET PARAMETER DUMPOK OR !
   IF PKIND = 2 THEN GOTO 40;               ! RELEASE OR NOTHING !
   IF PTEXT <> DMPOK THEN IF PTEXT <> NOREK THEN SYNTAX;
   IF PTEXT = DMPOK THEN RELEASE := 1;
   IF PTEXT = NOREK THEN RELEASE := 2;
   GTPM(COMLINE,PTEXT,PVALUE,PKIND,PSEP);   ! GET PARAMETER DUMPOK OR !
   IF PKIND = 2 THEN GOTO 40;               ! RELEASE OR NOTHING !
   IF PKIND <> 1 THEN SYNTAX;
   IF PTEXT <> DMPOK THEN IF PTEXT <> NOREK THEN SYNTAX;
   IF PTEXT = DMPOK THEN RELEASE := RELEASE + 1;
   IF PTEXT = NOREK THEN RELEASE := RELEASE + 2;
   IF RELEASE > 3 THEN SYNTAX;              ! IF RELEASE TWICE THEN SYNTAX !
   ! IF DUMPOK TWICE THEN SYNTAX !
   IF RELEASE = 2 THEN IF PTEXT = DMPOK THEN SYNTAX;
40:   IF PSEP <> 2 THEN SYNTAX;
   CHECKKIND;                                ! CHECK THAT JOB IS A JOB
                                               OR BATCH IS A BATCH !
   RTEXT1 := 0;
   IF ITEM2 <> NEW THEN IF ITEM2 <> OLD THEN SYNTAX;
   HEADER;                                   ! CHECK LABEL ON MAGTAPE !
   IF ITEM2 = NEW THEN GOTO 50;
   CLOSE(MT0,0);
   OPEN(MT0,3);
   SETPOSITION(MT0,2,32000);                ! THIS SETPOSITION WILL CAUSES IN
                                              A CALL OF PROCEDURE MTERROR
                                              WITH POSITION ERROR. MT0.ZBLOCK
                                              WILL CONTAIN NUMBER OF BLOCKS
                                              ON THE MAGTAPE + 1 !
   BLOCK := MT0.ZBLOCK;
50:   X := 3;
   CONNEC(LPT,X,LNAME);  ERRLPT := 0;       ! OPEN FOR PRINTOUT FILE !
   OUTTEXT(LPT,FF);                         ! MAKE A TOP OF FORM !
   FLAG := 1;                               ! REMEMBER THAT SOMETHING HAS
                                              BEEN WRITTEN IN PRINTOUT !
   IF COM = JOB THEN BEGIN MOVE(HJOB,0,HEADLINE,33,5);
               MOVE(JOBNAME,0,ITEM1,0,5);
               END;
   IF COM = HBATCH THEN BEGIN MOVE(HBATCH,0,HEADLINE,33,5);
               MOVE(NAME,0,ITEM1,0,5);
               END;
```

```
    OUTNAME;                                              ! CHANGE NULLS TO SPACES !
    MOVE(ITEM2,0,HEADLINE,39,5);
    OUTTEXT(LPT,HEADLINE);                                ! WRITE HEADLINE !
    IF COM = HBATCH THEN                                  ! ONE BATCH TO BE DUMPED !
      BEGIN
        DUMPBATCH;
        GOTO 100;
      END;
    BATCH.ZNAME := JOBNAME;
    OPEN(BATCH,1);  ERRBAT := 0;                          ! OPEN FOR JOB TO DUMP !
    SETPOSITION(BATCH,0,0);
    LENGTH := 4;                                          ! LENGTH OF BATCHNAME+STATUS=4 !
    RES := 0;  SUM := 0;
    WHILE RES <> 1 DO BEGIN
      ! RES IS SET TO 2 WHEN A NAME FOUND IN THE JOB BUT DOES
        NOT EXIST AS A DISC-FILE OR THE BATCH IS USED
        BY ANOTHER KEYSTATION. THEN THE DISC-ZONE IS NOT DESTROYED
        AND THIS MUST BE TOLD TO PROCEDURE GETNEX !
      IF RES = 2 THEN RES := 1;
      BATCH.ZNAME := JOBNAME;                             ! MAKES SURE THAT ZNAME IS JOBNAME
      GETNEX(BATCH,LENGTH,NAME,EWORD,RES);                ! GET NEXT BATCHNAME !
      IF RES = 1 THEN GOTO 70;                            ! IF END OF JOB THEN STOP !
      DUMPBATCH;
      SUM := SUM + DUMPNO;
 70:
    END;
    CLOSE(BATCH,1);  ERRJAT := 4;                         ! CLOSE JOB FILE !
    BINDEC(SUM,DECNO);                                    ! WRITE NO OF DUMPED RECORDS !
    OUTTEXT(LPT,TXTOTAL);
    OUTTEXT(LPT,DECNO);
    OUTTEXT(LPT,NL);



 100:
    OUTTEXT(LPT,TXEND);
    ! NORMALLY RETURN !
    RSPEC := 0;                                           ! NO SPECIAL ACTION !
    IF RTEXT1 = 0 THEN RTEXT1 := 12;                      ! OK !
    RTEXT2 := 0;                                          ! NO TEXT2 !
    RCON   := 0;                                          ! DO NOT CONTINUE !
    RESULT := 0;

 200:
    ! RELEASE DRIVER AND AREA PROCESSES !
    WHILE ERRMTO < 4 DO
        BEGIN  CLOSE(MTO,1);  ERRMTO := 4  END;
    WHILE ERRBAT < 4 DO
        BEGIN  CLOSE(BATCH,1);  ERRBAT := 4  END;
    WHILE ERRJAT < 4 DO
      BEGIN BATCH.ZNAME := JOBNAME;  CLOSE(BATCH,1);  ERRJAT := 4;  END;
    ! IF SOMETHING HAS BEEN WRITTEN IN THE PRINTOUT THEN
      WRITE AN END MEDIUM CHARACTER !
    WHILE ERRLPT < 4 DO BEGIN  IF FLAG = 1 THEN OUTTEXT(LPT,EM);
         CLOSE(LPT,1);  ERRLPT := 4  END;
 300:
    ! PACK PARAMETER RTEXT1 FOR RETURN !
    RTEXT1 := RTEXT1 SHIFT 10 + RTEXT2 SHIFT 4
             + RSPEC SHIFT 1 + RCON ;
    ! IF PRINTOUT IS TO BE OUTPUT ON HARDCOPY DEVICE OR DISPLAY SCREEN
      THEN SET THE "PRINTOUT-BIT"(BIT 13) IN RTEXT1 !
    IF FLAG = 1 THEN RTEXT1 := RTEXT1 + 2 SHIFT 1;
     TEXTMODE := 0;
    RETUR(PRETUR,RESULT,RTEXT1,TEXTMODE,COMLINE);                ! RETURN !
 END;
```

APPENDIX H: REFERENCES

1. MUSIL Programming Guide, RCSL: 42-i 0344

2. Data Entry, Release 2, Users Guide, part 2
RCSL: 43-GL 4796

APPENDIX I

INDEX

The references are pointing out page numbers.