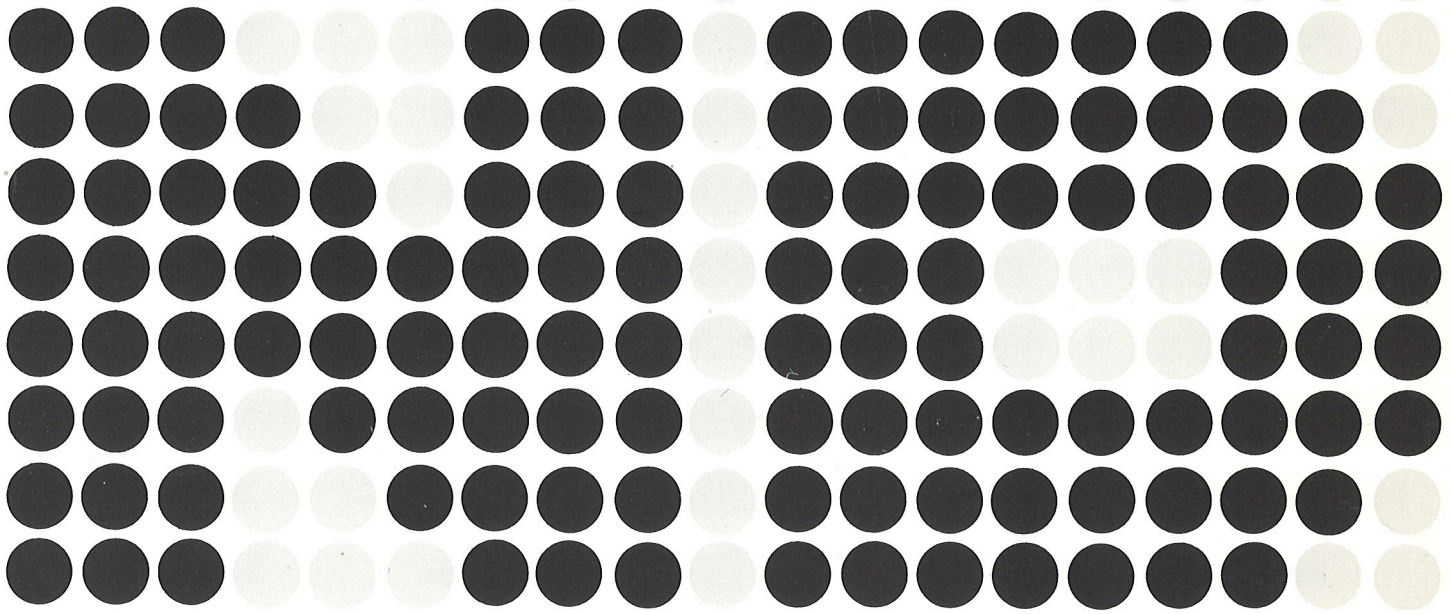
NOTIS-TF
Macro Guide

ND-63.009.01

Norsk Data



NOTIS-TF
Macro Guide

ND-63.009.01

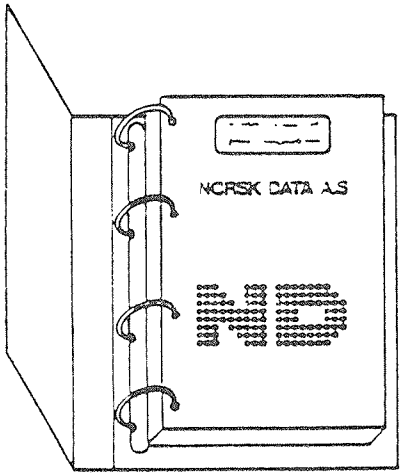
NOTICE

The information in this document is subject to change without notice. Norsk Data A.S assumes no responsibility for any errors that may appear in this document. Norsk Data A.S assumes no responsibility for the use or reliability of its software on equipment that is not furnished or supported by Norsk Data A.S.

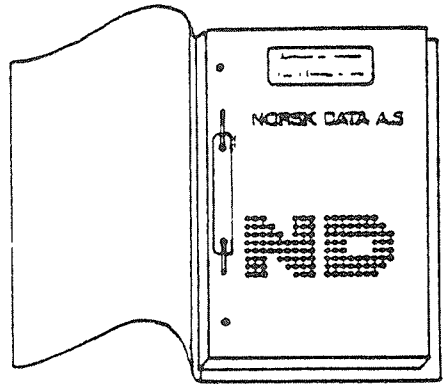
This manual is in loose leaf form for ease of updating. Old pages may be removed and new pages easily inserted if the manual is revised.

The loose leaf form also allows you to place the manual in a ring binder (A) for greater protection and convenience of use. Ring binders with 4 rings corresponding to the holes in the manual may be ordered in two widths, 30 mm and 40 mm. Use the order form below.

The manual may also be placed in a plastic cover (B). This cover is more suitable for manuals of less than 100 pages than for large manuals. Plastic covers may also be ordered below.



A Ring Binder



B Plastic Cover

Please send your order to the local ND office or (in Norway) to:

Documentation Department
Norsk Data A.S
P.O. Box 4, Lindeberg gård
Oslo 10

ORDER FORM

I would like to order

..... Ring Binders, 30 mm, at nkr 20,- per binder

..... Ring Binders, 40 mm, at nkr 25,- per binder

..... Plastic Covers at nkr 10,- per cover

Name .....
Company .....
Address .....
.....
City .....

PRINTING RECORD

Printing	Notes
12/83	Version 01

NOTIS-TF Macro Guide
Publ. No. ND-63.009.01
December 1983

NORSK DATA A.S
P.O. Box 4, Lindeberg gård
Oslo 10, Norway

Manuals can be updated in two ways, new versions and revisions. New versions consist of a complete new manual which replaces the old manual. New versions incorporate all revisions since the previous version. Revisions consist of one or more single pages to be merged into the manual by the user, each revised page being listed on the new printing record sent out with the revision. The old printing record should be replaced by the new one.

New versions and revisions are announced in the ND Bulletin and can be ordered as described below.

The reader's comments form at the back of this manual can be used both to report errors in the manual and to give an evaluation of the manual. Both detailed and general comments are welcome.

These forms, together with all types of inquiry and requests for documentation should be sent to the local ND office or (in Norway) to:

Documentation Department
Norsk Data A.S
P.O. Box 4, Lindeberg gård
Oslo 10

Preface:

THE PRODUCT

This guide is intended as a complete documentation of the use of macros in the ND text formatting system

NOTIS-TF SUT 10079J

THE READER

The guide is primarily intended for experienced users of the text processing system NOTIS-WP, and the text formatting system NOTIS-TF. The guide will be of particular interest to users who wish to create their own macros adapted to their needs for special document formats.

PREREQUISITE KNOWLEDGE

It is assumed that the user has some experience with the text formatting system NOTIS-TF, and is acquainted with the directives and document macros described in the NOTIS-TF Reference Manual - Text Formatter.

THE GUIDE

The guide provides a theoretical description of the construction and mode of operation of the various macro types in NOTIS-TF. The guide explains how these macros may be defined, used and deleted, and gives numerous examples of how they work in practice. Chapter 10 is entirely devoted to explaining the examples, most of which have been taken from the official macro library in NOTIS-TF.

RELATED MANUALS

NOTIS-TF Reference Manual - Text Formatter, ND-63.007, describes standard and advanced directives in the text formatting system NOTIS-TF.

NOTIS-WP Reference Manual - Editor, ND-63.002, is devoted to the text processing system NOTIS-WP.

TABLE OF CONTENTS

Section Page

Section Page

1 INTRODUCTION

1.1 WHAT IS A MACRO?

A macro is a combination of commands/directives, which can be defined to simplify trivial and time-consuming routines. Each time you wish to carry out a specific routine, you enter the name of the defined macro in your text document. This macro is *called/expanded*during formatting, and executes the routine automatically. The macro is uniquely identified (given a name). The definition is either entered in the text document, or included in a separate macro library. NOTIS-TF has its own, standard macro library which carries out numerous tasks.

The difference between a macro and a directive in NOTIS-TF is that most macros can be defined, altered and deleted by you. This enables you to to build up a well adapted macro library of your own. Directives, however, are defined as a part of the system, and cannot be altered by the user. The two types are identical in use, ie., they are entered into the text document and executed (expanded) when the document is formatted.

A macro often carries out the work of several directives, and can be used for many different purposes.

1.2 WHEN CAN MACROS BE USED?

Macros can be used for a number of things. We shall only mention a few examples, and leave it to your imagination and requirements to find others.

1) If a certain item of text, eg., a long word or expression, is repeatedly used throughout your text, you may define a macro for this expression. You then call this macro each time you want the expression included, thus saving time and making sure that the expression is always correctly spelled.

2) If a specific sequence of commands and directives and/or text is frequently used in the text, you may define a macro that executes the sequence. Such a macro may considerably reduce the amount of actual typing.

3) You may use macros to simplify the updating of blocks of text that change through the document, eg., version numbers in program documentation.

4) Macros may be defined to describe formats frequently used in a document or in a series of documents. In this way you may change the layout of the text each time you use a macro, thus greatly facilitating the use of standard formats.

5) You can use macros for automatic numbering of text items such as paragraphs. In this way, new paragraphs can be inserted without need for manual re-numbering of existing paragraphs. The existing chapter and section macros in NOTIS-TF have been defined for this purpose.

6) Macros can be used to create references in a document. The document may subsequently be modified, but the references will still be correct.

These are a few simple examples of the use of macros. More concrete examples will be given later.

1.3 DEFINING AND CALLING MACROS

It is important to clearly understand the difference between a macro call and a macro definition:

- A *macro definition* is a specification of what is to happen each time a macro is called. It therefore appears only once, and is either entered into the text document or included as part of a macro library.

- A *macro call* is entered in the text each time you want the specific macro definition to be executed. A macro call can thus be used several times in the text. During formatting of the document, NOTIS-TF will execute or *expand* the macro call in accordance with the definition.

Let us illustrate this with a small example:

Ex-1: *You are writing a long report on* **chloramphenicol**. *This is a boring word to write over and over again, and besides it is easy to misspell it. You therefore define a macro for this word, and enter the macro definition at the beginning of your text:*

`md/chl/chloramphenicol;

You have now defined a macro called **chl**. *Each time you now need to use the word chloramphenicol in the text, you call the macro by typing in* **`chl;**, *as in the following phrase:*

Extensive research has shown that `chl; is a good cure for .....

After formatting, the text will read:

Extensive research has shown that chloramphenicol is a good cure for .....

Many macros are defined with *parameters*, just as directives may have parameters. A parameter is a value that must be given in addition to the macro name, whenever the macro is called.

As you will see further on in this guide, certain macro definitions may be rather long and cover several lines. It is therefore a good principle to start each new line with a circumflex (^). Blank spaces at the beginning of a line may give unexpected results.

1.4 GLOSSARY

This section contains a list of words and expressions that have been frequently used in this guide, and their definitions. You may wish to refer back to the list whenever you encounter one of these expressions in the text, and do not remember the exact definition.

macro	A defined combination of commands/directives and/or text, which is uniquely identified. A macro may be defined and altered by the user. It is called in the same way as a directive in NOTIS-TF, and executes specific operations.
macro definition	A specification of the operations to be executed each time the macro is called.
macro call	An expression included in the text each time you want the macro to be expanded. Has the same format as a directive in NOTIS-TF, and is used in the same way.
macro expansion	The process resulting from a macro call when the text document is formatted. A macro expansion produces a result which corresponds to the macro definition.
level	The expansion of a macro may take place at various levels. This is the case whenever a macro call contains calls to other macros, or when directive quotes have been used.
directive quotes	Directive quotes are used to prevent the expansion of macro calls, and also decide the level on which the macro expansion is to take place at all times.
user macro	A macro type used to define special formats and layout in text documents.
system macro	A pre-defined macro type in the system. Cannot be influenced by the user. Is used to obtain access to fixed values in the system.

integer macro — A macro type containing an integer value or an arithmetic expression. This macro type is used for different ways of counting, for instance to number chapters and their related sections.

reference macro — A macro type used for references within a document.

trigger macro — A macro type which can only be defined, and not called. It is automatically expanded each time the defined conditions are fulfilled.

parameter — Macros may be defined to contain parameters. Such parameters must be included in the macro call. The number of parameters in the macro call must be the same as the number defined in the macro definition.

1.5 VARIOUS MACRO TYPES

NOTIS-TF operates with 6 different macro types;

	Definition	Call
USER MACROS	^MD/NAME/BODY;	^NAME/PARAMETER;
INTEGER MACROS	^IM/NAME/VALUE;	^$NAME/TYPE;
SYSTEM MACROS		^$NAME/TYPE;
REFERENCE MACROS	^RD/NAME/BODY;	^#NAME/SIZE;
TRIGGER MACRO	^TM/NAME/CONDITION/BODY;	
	^ST/NAME/CONDITION/BODY;	
TRIGGER STRINGS	^NAME/BODY;	

1.5.1 USER MACROS (MD)

The most common macro type is the user macro. User macros are used to define special formats and layout in a text document. The macro in the example on page 2 is a user macro. User macros can be defined to contain a combination of directives and/or text.

The user macro consists of two parts: *name* and *body*, and is defined in the following way:

```
`MD/Name/Body;
```

The *name* can be a combination of alpha characters, digits and the sign '-' (hyphen). However, a name has to start with an alpha character, and end with an alpha character or a digit.

The *body* can be a combination of text and directives/macros.

The macro is called/expanded in the following way:

```
`Name/parameter-1/parameter-2/... ../parameter-n;
```

The *parameters* are optional values that you have to input together with the name. The number of parameters you have to give after the macro name is defined in the body of the macro definition. Macros are often defined without parameters. This was the case with the *chl* macro on page 2.

Chapter 4 provides a full explanation of user macros.

1.5.2 INTEGER MACROS (IM)

The integer macro is a macro containing an integer value between -32768 and 32767. This macro type is used for different ways of counting, and to store numbers in macros that may in turn be called from other macros. Examples were numbered in this guide by using integer macros.

The integer macro consists of two parts: *name* and *value*, and is defined as follows:

```
^IM/name/value;
```

The *name*, the same rules apply as in user macros.

The *value* may be an integer value, or an arithmetic expression consisting of integer values.

The macro is called/expanded in the following way:

```
^$name/type;
```

Type may have the values:

A : Alphabetical representation in upper case.

a : Alphabetical representation in lower case.

R : Represented in Roman numerals, upper case.

r : Represented in Roman numerals, lower case.

N : Represented in numerics.

n : Represented in numerics.

Chapter 6 provides a full explanation of integer macros.

1.5.3 SYSTEM MACROS

The system macro is a macro type pre-defined in the system, and cannot be influenced by the user. These macros are used to obtain access to a series of so-called system values, such as the current date, current page number, line number, chapter number, etc.

The macro is called/expanded in the following way:

^*$name/type;*

Type may have the values:

A : Alphabetical representation in upper case.

a : Alphabetical representation in lower case.

R : Represented in Roman numerals, upper case.

r : Represented in Roman numerals, lower case.

N : Represented in numerics.

n : Represented in numerics.

Chapter 3 provides a full explanation of system macros.

1.5.4 REFERENCE MACROS (RD)

Reference macros are comparable to user macros, the difference being that a reference macro may be called before it has been defined. Reference macros can be used for references within a document, including forward references to parts of the document that have not yet been written.

The reference macro consists of two parts: *name* and *body*, and is defined as follows:

```
^RD/name/body;
```

The same rules apply for *name* and *body* as in user macros.

The reference macro is called/expanded in the following way:

```
^#name/size;
```

The *size* is the length/size needed to make room for the text resulting from the expanded macro. This size is actually only required in cases where the macro is called before it has been defined. However, it is a good principle to always include it. In cases where the size reserved proves to be larger than necessary, the text will be right justified in the reserved field. If the size is too small, the line the text is on will be expanded by the required number of positions.

Chapter 5 provides a full explanation of reference macros.

1.5.5 TRIGGER MACROS (TM, ST)

The trigger macro is a complicated kind of user macro, and is not always easy to understand and to use.

A trigger macro consists of four parts: *name*, *condition*, *body* and *priority*. It is defined as follows:

```
^TM/name/condition/body/priority;
or
^ST/name/condition/body/priority;
```

The same rules apply for *name* as in user macros.

Condition is a logical expression, which is explained in more detail in section 7.1

The same rules apply for the *body* as in user macros, with the exception that it is impossible to include parameters in a trigger macro.

Priority is explained in section 7.1

A trigger macro is not called. It is automatically expanded when the conditions defined are fulfilled. In few words, this means that as soon as a situation arises that fulfills the condition(s) defined, the macro is expanded. It is subsequently expanded again and again, until the conditions are no longer fulfilled.

Chapter 7 provides a full explanation of trigger macros.

Single-triggers are a special kind of trigger macro. They are expanded in the same way as trigger macros, but delete themselves after they have been expanded the first time. This means that a single-trigger is never expanded more than once.

1.5.6 TRIGGER STRINGS

Trigger strings are a pre-defined kind of single-trigger. The body may be defined by the user, while the name, condition and priority are pre-defined.

Each trigger string has a corresponding system macro, ie., a *flag* which indicates whether the string is being used or not.

A trigger string body is defined as follows:

^*name/body;*

where *name* is the pre-defined macro name.

The same rules apply for *body* as in trigger macros.

Chapter 8 provides a full explanation of trigger strings.

2 AN EXPLANATION OF THE SYNTAX

2.1 GENERAL SYNTAX FOR MACRO CALLS

The general syntax for macro calls is the same as for normal directives:

1) The call starts with the directive start sign. Default is the circumflex (^).(This sign may be redefined with the ^DS; directive). In integer macros this is followed by the dollar sign ($), and in reference macros by the hash (#).

2) Next comes the macro name, followed by a separating sign. Any non-alphanumeric character, except the hyphen (-) or the directive end sign, may be used as separating sign. But, you *must* use the *same separating sign* between all the parameters in a call. This means that the first non-alphanumeric character that is neither a hyphen nor the directive end sign becomes the separating sign in the macro call.

3) The various parameters, if any, are input after the macro name with separating signs between them. The number of parameters must be the same as in the definition.

4) A macro call is terminated with the directive end sign. Default is semicolon (;). (This sign may be redefined with the ^DE; directive).

Note that if you input a directive start sign followed by a space (blank), the directive start sign will be printed as text, and the space will be eliminated.

Ex-2: ^ *SURNAME;*

In the above example the macro *SURNAME* will not be called, but will be printed as ^*SURNAME;* in the text.

2.2 CALL LEVELS

It is important to understand the *level* principle in macro calls and macro definitions.

A call starts with a directive start sign and ends with a directive end sign *on the same level*.

Ex-3: *ˆMD/NAME/Peterson;*

Name : NAME
Body : Peterson
Call : ˆNAME;
Result : Peterson

This macro is defined with only one level, and the *elements* of the definition are *NAME* and *Peterson*. This means that a user macro is defined with the name *NAME* and the body *Peterson*.

It is often useful to include calls to other macros in a macro call definition. In the example below, the result in the call to the macro *ˆNAME* depends upon the macro *ˆSURNAME*.

Ex-4: *ˆMD/SURNAME/Smith;*
ˆMD/NAME/His name was ˆSURNAME;;

Name : NAME
Body : His name was Smith
Call : ˆNAME;
Result : His name was Smith

When NOTIS-TF reads the macro definition, the system will encounter a new directive start sign before the directive end sign, and a call to *ˆSURNAME;* which is one level higher will be initiated.

The above definition thus contains two calls that are on different levels. NOTIS-TF will treat these two calls in the following way:

The definition ˆMD will be started first, but when the system reaches the next directive start sign it will start the new call *ˆSURNAME;*

The reading of ˆMD will now be "put aside", and *ˆSURNAME;* will be expanded. Since this call is not on the lowest level (1), the text resulting from *ˆSURNAME;* will be written in as part of the body of the *ˆNAME;* call. When the *ˆSURNAME;* call has been terminated, the call on the lowest level (the definition, ˆMD) will be fetched out again and terminated.

In the above example the body in the definition contains a new call to a new macro. This macro may well in turn contain a call to yet another macro, which again contains another call, etc. In this way you may nest calls on an indefinite number of levels (provided you keep a clear head!).

As can be seen from the above example, the call on the upper level will *always* be executed first. You may compare a macro expansion to an arithmetic expression: the inner parenthesis is always calculated first.

A macro definition must always be balanced, ie., it must contain the same number of directive start/directive end signs and the same number of start quote/end quote signs (see the next section).

2.3 DIRECTIVE QUOTES

In NOTIS-TF the sequences '^<' and '^>' are considered as *start quote* and *end quote* signs respectively. In short, these quotes are used to prevent the expansion of a call sequence. This is often a requirement, for example when you define macros that require parameters.

Let us look at some examples:

```
Ex-5: ^MD/SURNAME/Smith;
      ^MD/NAME1/^SURNAME;;

      Name   : NAME1
      Body   : Smith
      Call   : ^NAME1;
      Result: Smith
```

This example is similar to the one we have used before, example 4. We have a macro definition with a new macro call in the body. The definition is processed as follows:

NOTIS-TF reads the definition until it encounters the new directive sign in the macro call ^*SURNAME;*. The first call is then interrupted and ^*SURNAME;* is executed. Thereafter the result is entered into the body of the definition, and the macro *NAME1* is defined. Here the result will be *Smith* each time the macro is called.

Now let us see what happens when you use quotes around the macro call in the body:

```
Ex-6: ^MD/SURNAME/Smith;
      ^MD/NAME2/^<^SURNAME;^>;

      Name: NAME2
      Body: ^SURNAME;
      Call: ^NAME2;
      Result: Smith
```

As you can see, the final result is the same as in the previous example. Nevertheless, the two are processed in different ways, and it is important that you understand this. The current macro definition is executed as follows:

NOTIS-TF reads until it encounters the first start quote sign. All subsequent signs encountered up to the corresponding end quote *are considered as ordinary text*. This means that calls which may be encountered between the two quotes also will be considered as ordinary text, and therefore not expanded right away. This is also true for quotes within quotes.

The macro stem is thus *ˆSURNAME;*. Each time the macro *ˆNAME2; is called, ˆSURNAME;* is expanded. The result of *ˆNAME2;* will at all times depend upon the definition of *ˆSURNAME;*.

Let us illustrate the difference between the two types with another example:

Ex-7: *We are now in a situation where the call* **ˆNAME1;** *results in* **Smith** *and* **ˆNAME2;** *also results in* **Smith**.

But what happens if we change the definition of **ˆSURNAME;**?

ˆMD/SURNAME/Brown;

The kall ˆNAME1; still results in Smith. The call ˆNAME2; now results in Brown

Thus, the difference between the two methods is that when a macro call is written **without** *quotes the result is irrevocably defined at the time of definition, whereas the result of a macro call written* **with** *quotes is at all times dependent upon the valid definition of ˆSURNAME;*

It is important to note that the quotes themselves disappear after the macro has been expanded.

In the above example this means that *ˆSURNAME;* and not *ˆ<ˆSURNAME;ˆ>* becomes the body of the macro *NAME2*.

When quotes are used in macro definitions, they do not necessarily have to enclose the whole body. This is illustrated in the following example:

```
Ex-8: ^MD/SURNAME/BROWN;
      ^MD/FIRST-NAME/John;
      ^MD/MIDDLE-NAME/T. ;
      ^MD/NAME2/^<^FIRST-NAME;^> ^MIDDLE-NAME;^<^SURNAME;^>;


      Name   : NAME2
      Body   : FIRST-NAME; T. ^SURNAME;
      Call   : ^NAME2;
      Result: JohnT. Brown
```

In this case, the expansion of *FIRST-NAME* and *SURNAME* will be delayed because these are written between quotes, whereas *MIDDLE-NAME* will be expanded when *NAME2* is defined. The same technique can be used if you want to include the values of system or integer macros at the time of definition.

Let us look at an example of a macro requiring a parameter when it is called:

```
Ex-9: ^MD/TITLE/^<Manual for ^1;^>;

      Name   : TITLE
      Body   : Manual for ^1;
      Call   : ^TITLE/NOTIS-TF;
      Result: Manual for NOTIS-TF
```

The call ^1; is a special call which means that a parameter is to be inserted here. As you can see, it was important that the call ^1; was not executed at the time of definition, but remained in the body. ^1; was undefined when ^TITLE; was defined. The result of the macro thus depends upon the parameter given when the macro is called. More about parameters on page 24.

On page 24 and page 27 you will find examples of practical use of quotes, and you will see that quotes are of importance when you define more advanced macros.

Each time a start quote is encountered, the execution will continue one level higher. The execution of this level does not terminate until the corresponding end quote has been encountered. Start quotes within quotes lead to the execution going one level higher.

Macro definitions must be balanced, ie., there must be an equal number of start quotes and end quotes.

3 DETAILED DESCRIPTION OF THE MACRO TYPE SYSTEM MACRO

This is the simplest macro type: you can neither define nor modify it, but only call it.

A system macro is called in the following way:

```
'$name/type;
```

Below is a list of all system macros, with a short functional description:

CALL	DESCRIPTIION	FORMAT
$PN	Current page number	numeric value
$CN	Current chapter number	numeric value
$SN	Current section number	numeric string
$AN	Current appendix number	numeric value
$CPOS	Current line position	numeric value
$CLINE	Current line number	numeric value
$SECLEV	Current section level	numeric value
$LM	Left margin	number of char.
$RM	Right margin	number of char.
$LB	Left border	number of char.
$OB	Right border	number of char.
$TB	Top border	number of lines
$BB	Bottom border	number of lines
$PL	Page length	number of lines
$PW	Page width	number of char.
$TW	Text width = $PW - $LB - $OB	number of char.
$SI	Section indentation	number of char.
$SS	Section spacing	number of lines
$SF	Minimum section size	number of lines
$PI	Paragraph indentation	number of char.
$PS	Paragraph spacing	number of lines
$PF	Minimum paragraph size	number of lines
$BT	Bold text level	numeric value
$BS	Bold section level	numeric value
$LS	Line spacing	number of lines
$HP	Horizontal pitch	char/inch
$VP	Vertical pitch	lines/inch
$TI	Title	text
$AU	Author	text
$DI	Distributiion list	text
$TO	Addressee	text
$LH	Letter head	text
$RF	Reference	text
$AS	Abstract	text
$CH	Chapter heading	text
$H1	Page header 1	text
$H2	Page header 2	text
$TL	Page trailer	text

$CHEAD	Table of contents heading	text
$DX	Flag for duplex copying	active if > 0
$F-PH-STR	Flag for PH-STR	active if > 0
$F-TL-STR	Flag for TL-STR	active if > 0
$F-EOD-STR	Flag for EOD-STR	active if > 0
$F-EOF-STR	Flag for EOF-STR	active if > 0
$F-PAR-STR	Flag for PAR-STR	active if > 0
$F-UND-STR	Flag for UND-STR	active if > 0
$DATE	Date	20.07.1983
$YEAR	Full year	1983
$YR	Partial year	83
$MM	Month	07
$M	Single month	7
$DD	Day	02
$D	Single day	2
$FDATE	File date	20.07.1983
$FYEAR	File full year	1983
$FYR	File partial year	83
$FMM	File month	07
$FSM	File single year	7
$FDD	File day	02
$FSD	File single day	2
$TIME	Time (English)	12:45 am.
$TID	Time (24 hour clock)	00.45
$HOUR	Hour	00
$MIN	Minute	45
$SEC	Second	36

All these system macros may have a parameter in the call. This parameter may have one of the following values:

A : Alphabetical representation in upper case.

a : Alphabetical representation in lower case.

R : Represented in Roman numerals, upper case.

r : Represented in Roman numerals, lower case.

N : Represented in numerics.

n : Represented in numerics.

Numeric value is default. The representation in Roman numerals is limited upward to 3999, and Alphanumeric representation to 18278.

Alphabetical representation functions in the following way:

1=a, 2=b, 3=c,.....26=z
27=aa, 28=ab, 29=ac.....52=az
53=ba, 54=bb, 55=bc, etc.

Ex-10: *If you write in your text:*

This is page ^$PN;, *the result will be:*

This is page 19, *where* **19** *is the number of the current page in the formatted document.*

If you write in your text:

This is page ^$PN,a;, *the result will be:*

This is page s, *where* **s** *is the number of the current page in the formatted document.*

If you write in your text:

This is page ^$PN,R;, *the result will be:*

This is page XIX, *where* **XIX** *is the number of the current page in the formatted document.*

However, the system macro $SN is a special case. This macro may have two parameters in the call, the first of which indicates the separator between numbers on different section levels.

Ex-11: *If you write:*

In this section, ^$SN,/;,...

the result will be:

In this section, section 3/1/2,...

The call **^$SN=)=a;** *gives the result* **c)a)b.**

The parameter for the numeric system thus becomes parameter two in this macro.

The call **^$SN;**, *without parameters, gives the default result* 3.1.2 *because the full stop is the default separating sign.*

4 DETAILED DESCRIPTION OF THE MACRO TYPE USER MACRO

There are some directives that are directly related to user macro processing:

MD : Makro Define, defines a user macro.

PM : Parameter Include, fetches a parameter to the call.

MK : Makro Kill, deletes a user macro.

MA : Makro Append, extends the definition of a user macro.

MI : Makro Insert, extends the definition of a user macro.

MR : Makro Remove, removes part of a user macro definition.

CM : Clear Makro, removes the whole user macro body.

DM : Dump Makro, writes the macro definition out on the screen.

FIX : "Fixes" (locks) a user macro definition, ie., it cannot be deleted.

NREDEF: No-Redefine, prevents redefining of a user macro.

4.1 MD, MACRO DEFINE

The directive is used to define a user macro. It is an important directive, and very frequently used.

A user macro functions as follows:

1) The macro first has to be *defined*. The definition must either be written in the text document, or included in a modified version of the standard macro library (file NOTIS-TF-ENG-xxx:LIB) which is a part of the system.

2) The *call* is then input in the text in the position where you want it. The macro is executed (expanded) when the document is formatted. In the formatted text the call is then substituted by the text resulting from the macro.

In other, and slightly more technical words: the text document read by NOTIS-TF is "pushed aside" each time a macro is encountered, and the macro body is read instead. Once the macro body has been read and executed, NOTIS-TF picks up the text document again, and the reading continues where it was interrupted, ie., after the macro call. You may therefore compare a user macro call to the inclusion of a new document, where the macro body is the new document.

A user macro consists of two part: the *name* and the *body*. It is defined in the following way:

```
'MD/name/body;
```

The name can be a combination of alpha characters, digits and the sign '-' (hyphen). However, a name has to start with an alpha character, and end with with an alpha character or a digit.

The body can be a combination of text and directives/macros.

4.1.1 USER MACRO WITH A PERMANENT BODY

The simplest form of user macro is a macro with a permanent body.

Ex-12: *ˆMD/PRODUCT/NOTIS-TF;*

Name: *PRODUCT*
Body: *NOTIS-TF*
Call: *This is a description of ˆPRODUCT;*
Result: *This is a description of NOTIS-TF*

As you can see, the macro call in the text document will be substituted by the macro body in the formatted document.

4.1.2 USER MACRO WITH MACRO CALL IN THE BODY

Ex-13: *ˆMD/TITLE/Manual for ˆPRODUCT;;*

Name: *TITLE*
Body: *Manual for NOTIS-TF*
Call: *ˆTITLE;*
Result: *TITLE*

The result of this macro will also be a constant text in the body, despite the body having a variable at the time of definition. That is to say that *PRODUCT* will be executed first, because the call for *PRODUCT* is inside the call for *MD*. This leads to the result of *PRODUCT* being entered into the body of *TITLE*, thus making it a constant text.

A macro call is not executed until the whole call has been read, ie., up to and including the directive end sign on the lowest level.

In this case the execution/expansion of *ˆTITLE;* is carried out as follows:

- The directive start sign starts the call.
- Then NOTIS-TF reads up to the first non-alphabetic, non-numeric sign, in this case ';'.
- NOTIS-TF subsequently goes to the definition of *TITLE* and continues the reading there.

The definition of *ˆTITLE;* will be carried out as follows:

- The directive start sign starts the call (the definition)..
- Then NOTIS-TF reads up to the first non-alphabetic, non-numeric sign, in this case '/'. What has been read until now is considered to be the name of the call.
- NOTIS-TF subsequently reads up to the next separator sign. In this case there is none, and the system therefore continues to read up to the directive end sign. However, before it reaches it a new call is encountered. The execution of the MD-call is therefore temporarily 'set aside', and NOTIS-TF moves up one level.
- The call for *PRODUCT* is then read in full, and executed. This leads to the result of *PRODUCT* being written where the call was found, ie., in the body of the MD-call. When the PRODUCT-call has been terminated, the system switches back to the previous level, ie., the MD-call.
- The reading is now picked up again where it was interrupted, ie., after the call for *PRODUCT*. This means that once the body of the MD-call has been fully read, it has been changed into a constant text.
- The body is then terminated by the directive end sign. The call is now fully read, and the execution may start.
- What is now to be executed is a call with the name *MD*, where the first parameter is *TITLE* and the body is *Manual for NOTIS-TF*.
- The body is then entered into the formatted text at the position where the call *ˆTITLE;* was input in the text document.

A call is thus read in its entirety before it is executed. This means that all calls on higher levels are executed before the main call is processed.

4.1.3 USER MACRO WITH A VARIABLE PART IN THE BODY - MACRO-CALL IN QUOTES

We shall go one step further and look at a macro with a variable part in the body.

Ex-14: *ˆMD/TITLE/ˆ<Manual for ˆPRODUCT;ˆ>;*

Name : *TITLE*
Body : *Manual for ˆPRODUCT;*
Call : *ˆTITLE;*
Result : *Manual for NOTIS-TF*

Using directive quotes causes the call for *PRODUCT* to remain in the body of the MD-call instead of being executed at the time of definition.

This can be explained as follows:

> When the system starts to read the body of the MD-call, it meets a start quote. This means that everything that is read up to the corresponding end quote is considered as normal text. The call for *PRODUCT* is therefore not executed, nor does *PRODUCT* need to have been defined at this point.

When *TITLE* is now called, you may imagine that the text document currently being read is 'put aside', and the body of *TITLE* considered as a text document. The body is now *Manual for ^PRODUCT;*. The call for *PRODUCT* will only be executed when *TITLE* is called. It will then be executed exactly as if it had been entered in the text document.

We say that the macro has a variable body because it is dependent upon other macros.

We can imagine that the macro *TITLE* is being used on various occasions as it is, while the definition of *PRODUCT* varies. The result of the macro can thus be modified without modifying the macro definition. The importance of directive quotes is thus evident, because we could not alter the definition of *PRODUCT* without them.

4.1.4 USER MACRO INCLUDING PARAMETERS

Another way of defining a macro with a variable body is to include parameters. This is a very useful mechanism, and is frequently used.

Ex-15: *^MD/TITLE/^<Manual for ^1;^>;*

```
Name   : TITLE
Body   : Manual for ^1;
Call   : ^TITLE/NOTIS-TF;
Result : Manual for NOTIS-TF
```

The call ^1; is a special directive, and means that parameter 1 which you write in the macro call is to be included here. In this manner you may include an 'infinite' number of parameters in a macro simply by numbering them successively. The parameters do not have to be defined in ascending order. The same parameter may be used several times in the same macro. This means that a parameter is not 'used up' after you have included it the first time.

The result of *TITLE* is now independent of other macros, but dependent upon the parameter given in the call for *TITLE*.

Remember that directive quotes are necessary to prevent ^1; from being executed at the time of definition. The parameter is not to be entered until the macro is called.

4.1.5 INCLUDING PARAMETERS WITH DEFAULT VALUES

When you used included parameters you may also define a default value for each parameter. This means that if you do not specify the parameter when the call is made, the default value will be used. This is done as in the example below:

Ex-16:

```
^MD/TITLE/^<Manual for ^1,NOTIS-WP;^>;

Name     : TITLE
Body     : Manual for ^1,NOTIS-WP;

Call-1   : ^TITLE/NOTIS-TF;
Result   : Manual for NOTIS-TF
```

In call-1 the parameter NOTIS-TF was given, and was thus included as parameter 1 according to the definition.

```
Call-2   : ^TITLE;
Result   : Manual for NOTIS-WP
```

In call-2 no parameter was given. The default value *NOTIS-WP* was therefore used.

```
Call-3   : ^TITLE/;
Result   : Manual for NOTIS-WP
```

In call-3 we gave what is known as an *empty parameter*. This also leads to the default value being used. It is necessary to use empty parameters in those cases where there is more than one parameter to a call, and you want to use default values for some of them. You give empty parameters by writing two separators one after the other. A parameter containing one or more blanks (spaces) will NOT be considered empty. See the next example.

As mentioned above you may include more than one parameter in a macro. This is illustrated in the following example.

```
Ex-17: ^MD/TITLE/^<Manual for ^1,NOTIS-WP; version ^2,A;^>;
       Name     : TITLE
       Body     : Manual for ^1,NOTIS-WP; version ^2,A;

       Call-1   : ^TITLE/NOTIS-TF/I;
       Result   : Manual for NOTIS-TF version I
```

In call-1 we gave the parameters *NOTIS-TF* and *I*. These were therefore included according to the definition.

```
       Call-2   : ^TITLE/NOTIS-TF;
       Result   : Manual for NOTIS-TF version A
```

In call-2 we omitted parameter 2. This caused the default value to be used.

```
       Call-3   : ^TITLE;
       Result   : Manual for NOTIS-WP version A
```

In call-3 we omitted both parameters. This caused both the default values to be used.

```
       Call-4   : ^TITLE//;
       Result   : Manual for NOTIS-WP version A
```

In call-4 we gave 2 empty parameters. This led to the same result as in call-3.

```
       Call-5   : ^TITLE//////////////;
       Result   : Manual for NOTIS-WP version A
```

In call-5 we gave 14 empty parameters. This also gave the same result as in call-3, illustrating that any excess parameters will simply be ignored.

```
       Call-6   : ^TITLE//G;
       Result   : Manual for NOTIS-WP version G
```

In call-6 we gave an empty parameter 1, whereas *G* was given as parameter 2. This caused the default value to be used for parameter 1, while parameter 2 was included according to the definition.

4.1.6 INCLUDING PARAMETERS WITH THE AID OF THE PM DIRECTIVE

There is also another directive for parameter inclusion: *PM*.

This directive may have 2 parameters: *Number* and *default value*.

```
ˆPM/number/default-value;
```

Number indicates the parameter number, and must be a positive numeric value.

The same rules apply for *default value* as for a user macro body.

In the following example we shall try to demonstrate the difference between these two ways of including parameters.

Ex-18:
```
ˆMD/TITLE/ˆ<Manual for ˆ1,NOTIS-WP; version ˆPM,2,A;ˆ>;
ˆMD/PRODUCT/NOTIS-TF;
ˆMD/VERSION/J;

Name     : TITLE
Body     : Manual for ˆ1,NOTIS-WP; version ˆPM,2,A;

Call-1   : ˆTITLE/NOTIS-TF/I;
Result   : Manual for NOTIS-TF version I

Call-2   : ˆTITLE/ˆPRODUCT;/ˆVERSION;;
Result   : Manual for NOTIS-TF version J

Call-3   : ˆTITLE/ˆ<ˆPRODUCT;ˆ>/ˆ<ˆVERSION;ˆ>;
Result   : Manual for NOTIS-TF version ˆVERSION;
```

In call-1 we gave a constant text in both parameters, and they were therefore both included in the same way.

In call-2 we gave two macro calls as parameters. However, these were not written in directive quotes, and were therefore executed when the call was read. The call was thus *executed* with two constant texts as parameters.

In call-3 we gave two macro calls written in quotes as parameters. As you can see, the parameters were in this case treated differently. It is in such cases that the difference between the two ways of including parameters becomes apparent. In the present case the result of parameter 2 was not a desirable one. However, we picked a simple example for the sole purpose of describing the two different methods.

The differences between the two methods are:

Parameter inclusion according to the first method with *ˆnumber/standard*; is execution by expanding the parameter as if it were a call. This means that the parameter is read as a body in a macro expansion, causing possible calls inside the parameter to be executed. As you will have seen in call-3, the call for the macro *PRODUCT* in parameter 1 was executed.

Parameter inclusion according to the second method, *ˆPM/number/standard;, is executed by the parameter being copied in, but not analyzed.* Possible calls in the parameter are therefore not executed. As you will have seen in call-3, the call for *VERSION* in parameter 2 was not executed. The PM directive has the same effect as an extra set of directive quotes. The execution of the macro definition will *go up one level* when the system encounters a ˆPM; directive.

In the following example we shall illustrate a more concrete case, where it may be desirable to include a parameter without expansion:

Ex-19: *ˆMD/MAC-DEF/ˆ<ˆMD,ˆ1;,ˆPM=2;;ˆ>;*

Name : *MAC-DEF*
Body : *ˆMD,ˆ1;,ˆPM=2;;*

Call-1 : *ˆMAC-DEF/INDENTATION/ˆ<ˆBL=2;ˆLM=+3;ˆ>;*
Result : *A macro will be defined under the name* **INDENTATION** *and with the body* **ˆBL=2;ˆLM=+3;**.

Call-2 : *ˆMAC-DEF/EX/ˆ<ˆNF;ˆLM=+7;ˆIP=Ex: ;ˆ>;*
Result : *A macro will be defined under the name* **EX** *and with the body* **ˆNF;ˆLM=+7;ˆIP=Ex: ;**.

The use of the PM directive makes it possible to include directives as part of a parameter.

In chapter 10 on page 80 under the example of the macro *FIG-CP*, you will find a further example of the use of the PM directive.

4.2 MK, MACRO KILL

This directive is used to delete or invalidate a macro, ie., to remove it from the set of defined macros. Attempting to call a *killed* macro leads to the same result as attempting to call a macro that has not been defined, namely an error message.

The MK directive demands a parameter, and is called as follows:

^MK/Name;

where *Name* is the name of the macro to be killed.

When you define a macro, it occupies space in the computer memory. How much space it occupies depends on its size. When you subsequently kill the macro, you liberate the space it occupied and make room for new macro definitions. If you work with numerous and large macros, it is therefore useful to occasionally clean up a little by killing macros which will no longer be used.

If you define a macro twice (or several times), ie., if you have two (or several) macro definitions under the same name, the last definition will be considered valid. However, the previous definition(s) will not disappear, but will simply be unavailable. If you now kill the macro bearing this name, you will kill the *last* definition and the former definition will again be valid. The following example may be an illustration:

Ex-20: *^MD/NAME/Smith;*

*The call ^***NAME***; will result in* **Smith**.

^MD/NAME/Jones;

*The call ^***NAME***; will now result in* **Jones**.

^MK/NAME;

*The call ^***NAME***; will now result in* **Smith**.

If you try to kill a macro that has not been defined, you will receive an error message.

4.3 MA. MACRO APPEND

The directive is used to extend the body of a user macro or a reference macro.

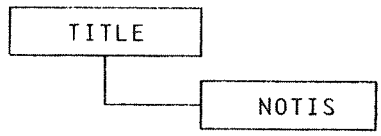
The MA directive demands at least two parameters, and is called as follows:

'MA/Name/body-element/../body-element;

where *Name* is the name of the macro to be extended, and *body-element* is the addition to the existing macro body. Of course the same syntax rules as for macro definition also apply here.

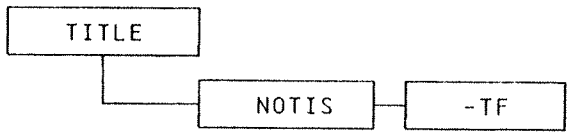
The logical construction of a user macro or a reference macro is outlined below, demonstrating how a macro can be extended.

The definition *'MD/TITLE/NOTIS;* leads to a macro being defined which is built up like this:



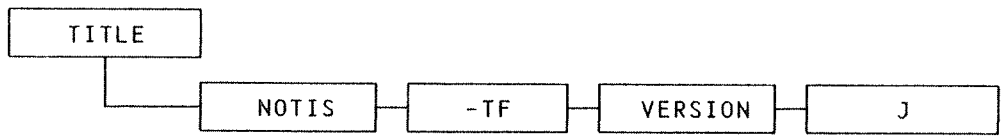
We then say that the macro consists of two *elements*, where the name is one element and the body another.

The call *'MA/TITLE/-TF;* will in this case lead to the macro being extended in the following way:



As can be seen in the syntax definition for this call, it is possible to extend a macro by more than one body-element in the same call. .

The call *^MA/TITLE/ VERSION/ J;* will in this case lead to the macro being extended thus:



4.4 MI. MACRO INSERT

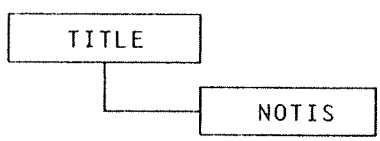
This directive is also used to extend the body of a user macro or reference macro (see also the MA directive, page 30).

The MI directive demands at least two parameters, and is called as follows:

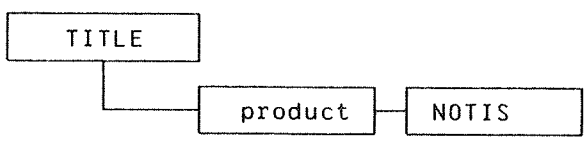
^MI/Name/body-element/../body-element;

where *Name* is the name of the macro to be extended, and *body-element* is what is to be included in the existing macro body. The same syntax rules as for macro definition of course also apply here.

The definition *^MD/TITLE/NOTIS;* causes a macro to be defined which is built up as follows:



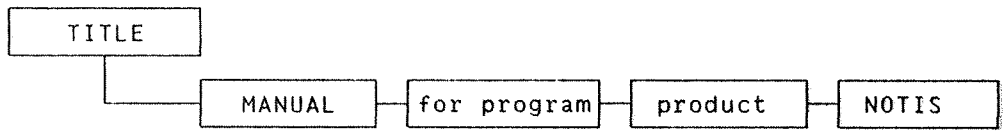
The call *^MI/TITLE/product;* will in this case cause the macro to be extended as follows:



The directive therefore in principle functions as the MA directive, with the exception that new body-elements are inserted before the existing elements.

As can be seen in the syntax definition for this call, it is possible to extend a macro by more than one body-element in the same call. .

The call *'MI/TITLE/ MANUAL/ for program;* will in this case cause the macro to be extended as follows:



4.5 MR. MACRO REMOVE

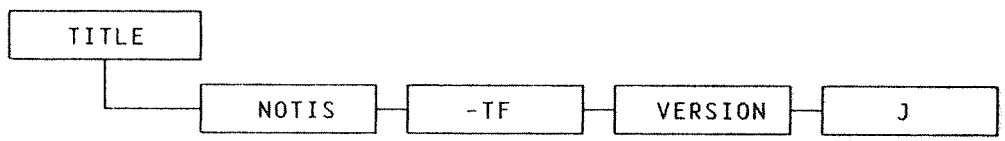
This directive is used to remove a part (an element) of the body in a user macro or a reference macro.

The MR directive is called in the following way:

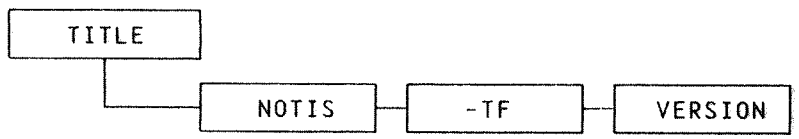
'MR/Name/direction;

Where *Name* is the name of the macro to be reduced. The parameter *direction*, which is optional, specifies how the macro body is to be reduced. It can be given one of the values F or B, where F indicates that the first (foremost) body-element is to be removed, whereas B (this is default) indicates that the body-element at the very back is to be removed.

To illustrate this, we can start with a macro looking like this:

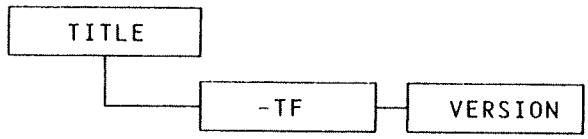


The call *'MR/TITLE;* will cause the macro to be reduced to:



In this call it is not specified how the macro is to be reduced. The default value B (back) is therefore used. This therefore caused the last element in the macro body to be removed.

The call *'MR/TITLE/F;* will cause the macro to be further reduced, thus:



As we can see, the foremost body-element was now removed.

4.6 CM. CLEAR MACRO

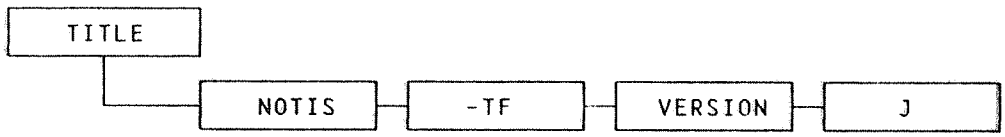
This directive is used to remove the whole body of a user macro or reference macro.

The CM directive is called in the following way:

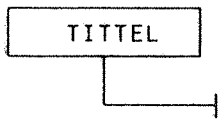
`CM/Name;

where *Name* is the name of the macro whose body is to be removed.

To illustrate this we may start with a macro looking like this:



The call `CM/TITLE; will cause the macro to look like this:



We can now use the MA directive to build a new body for the macro.

If you want to redefine a macro, the sequence CM, MA will often be quicker and less resource-consuming than the sequence MK, MD. This is in particular true when larger and more complex macros are concerned. The problem will of course not be very noticeable if redefinition occurs only once. However, for repetitive redefinitions, for instance inside a macro often used, the CM, MA sequence may reduce the resource consumption in NOTIS-TF somewhat.

4.7 DM, DUMP MACRO

This directive can be very helpful if you no longer remember exactly what you wanted your various macros to do. With the DM directive you may cause all the important data about a macro to be displayed on the screen at any time during the formatting process. While the data is displayed, the formatting stops and and the system waits for you to decide whether it should continue or not.

The DM directive is call in the following way:

^*DM/Name*;

where *Name* is the name of the macro to be dumped on the screen.

4.8 FIX. FIX MACRO

This directive fixes or 'freezes' a macro definition. This means that this particular version of the macro cannot be killed. However, if you define a new macro under the same name, the new definition will not be frozen.

The FIX directive is called in the following way:

^FIX/name/name/.../name;

where *Name* is the name of the macro to be fixed or frozen. As can be seen from the syntax description, it is possible to freeze several macros in one call.

4.9 NREDEF, SET NO-REDEFINE

With this directive you may decide that a macro is to be non-redefinable.

The NREDEF directive is called in the following way:

'NREDEF/name/name/.../name;

where *name* is the name of a macro which is to be non-redefinable. As can be seen from the syntax description, it is possible to set 'no-redefine' for several macros in one call.

The directive may be useful in a macro library to reserve macro names whose definitions are hidden from other users.

5 DETAILED DESCRIPTION OF THE MACRO TYPE REFERENCE MACRO

Certain directives are intimately linked to the processing of reference macros:

RD : Reference Define, defines a reference macro.

MK : Macro Kill, deletes a reference macro.

MA : Macro Append, extends the definition of a reference macro.

MI : Macro Insert, extends the definition of a reference macro.

MR : Macro Remove, removes part of a reference macro definition.

DM : Dump Macro, displays the macro definition on the screen.

FIX : Fixes a reference macro definition, cannot be deleted.

NREDEF: No-Redefine, prevents redefinition of a reference macro.

5.1 RD, REFERENCE DEFINE

This directive is used to define a reference macro.

The construction of a reference macro is directly comparable to that of a user macro. The only difference is that the reference macro may be called before it has been defined.

The syntax for the definition of a reference macro is therefore as follows:

```
^RD/Name/Body;
```

where the rules are the same as for user macros.

A reference macro is called in the following way:

```
^#Name/Size;
```

where the rules for name are the same as for user macros, and *Size* represents the number of character positions needed for the reference.

Ex-21: *Let us look at a reference macro for page number reference as an example. You are in the process of writing the introduction to a subject, and you want to include a reference to the page where this subject will later be described in depth. You may then, for example, input:*

this will be explained in depth on page ^#PAGE-REF/2;..

This call will reserve 2 character positions in the text at the location where the reference will in due time be included. **PAGE-REF** *is a name of your own invention, which you have to use again when you later define the macro.*

When you subsequently reach the location in the text where you give the in-depth description of your subject, you define the reference macro in this way:

^RD/PAGE-REF/^$PN;;

You are now giving your macro the name **PAGE-REF** *already decided upon in the macro call, and the body contains a call for the page number system macro. The macro is thus defined to contain the page number of the current page. Now matter how much you change document at a later stage, this page reference will always be correct. This page number will be right justified in the two character spaces that you reserved for it in the macro call.*

If it turns out that the page number needs, for instance, three spaces, the reference line will be expanded by one space and you will receive a message about that during formatting.

Since the construction of reference macros is almost identical to that of user macros, we shall not explain them further but refer you to the chapter devoted to user macros (see page 20).

5.1.1 MK. MACRO KILL

This directive can be used with reference macros in the same way as with user macros (see page 29).

5.1.2 MA. MACRO APPEND

This directive can be used with reference macros in the same way as with user macros (see page 30).

5.1.3 MI, MACRO INSERT

This directive can be used with reference macros in the same way as with user macros (see page 31).

5.1.4 MR

This directive can be used with reference macros in the same way as with user macros (see page 32).

5.1.5 CM, CLEAR MACRO

This directive can be used with reference macros in the same way as with user macros (see page 34).

5.1.6 DM, DUMP MACRO

This directive can be used with reference macros in the same way as with user macros (see page 35).

5.1.7 FIX, FIX MACRO

This directive can be used with reference macros in the same way as with user macros (see page 36).

5.1.8 NREDEF, SET NO-REDEFINE

This directive can be used with reference macros in the same way as with user macros (see page 37).

6 DETAILED DESCRIPTION OF THE MACRO TYPE INTEGER MACRO

Certain directives are directly attached to the processing of integer macros:

IM : Integer Macro Define, defines an integer macro.

MK : Macro kill, kills (deletes) an integer macro.

DM : Dump Macro, displays the macro definition on the screen.

FIX : Fixes an integer macro definition, cannot be deleted.

NREDEF: No-Redefine, prevents redefinition of the integer macro.

6.1 IM, INTEGER MACRO DEFINE

The directive is used to define an integer macro.

The integer macro consists of two parts: *Name* and *Value*, and is defined in the following way:

```
'IM/Name/Value;
```

The same rules apply to *Name* as to macro names in general.

The parameter *Value* may be omitted, in which case the value is undefined. When used, the parameter must consist either of a number alone, or of an arithmetic expression. Note that Roman numerals and alphabetically represented numbers cannot be used in arithmetic expressions.

The rules for arithmetic expressions are the same as for the AR-directive in NOTIS-TF. See section 9.2 on page 57.

The following arithmetic operators may be used: ( ) + - * /.

Other rules: The calculations are carried out from left to right, with parentheses. The number of parenthesis levels is practically unlimited.

The simplest integer macro definition is therefore:

Ex-22: *^IM/QUANTITY/100;*

```
Name    : QUANTITY
Value   : 100
Call    : ... the quantity being ^$QUANTITY;...
Result  : ... the quantity being 100...
```

The example below is a definition with an arithmetic expression, where the arithmetic expression is used to count relative to the integer macro in the previous example.

Ex-23: *^IM/QUANTITY-2/^$QUANTITY;+1;*

```
Name    : QUANTITY-2
Value   : ^$QUANTITY; + 1
Call    : ^$QUANTITY-2;
Result  : 101
```

In this manual we have used a combination of integer macros and other macro types to automatically number examples. In section 10.3 on page 70 we demonstrate how we have done this.

6.2 MK. MACRO KILL

This directive can be used with integer macros in the same way as with user macros (see page 29).

6.3 DM. DUMP MACRO

This directive can be used with integer macros in the same way as with user macros (see page 35).

6.4 FIX. FIX MACRO

This directive can be used with integer macros in the same way as with user macros (see page 36).

6.5 NREDEF. SET NO REDEFINE

This directive can be used with integer macros in the same way as with user macros (see page 37).

7 DETAILED DESCRIPTION OF THE MACRO TYPE TRIGGER MACRO

The trigger macro does not have to be called, but expands (is 'triggered') automatically when the conditions you have defined are fulfilled.

Certain directives are directly concerned with the processing of trigger macros:

TM : Trigger Macro Define, defines a trigger macro.

MK : Macro kill, kills (deletes) a trigger macro.

MA : Macro Append, extends the definition of a trigger macro.

MI : Macro Insert, extends the definition of a trigger macro.

MR : Macro Remove, removes part of the trigger macro definition.

DM : Dump Macro, displays the macro definition on the screen.

FIX : Fixes trigger macro definition so it cannot be deleted.

ENABLE : Activates a trigger macro.

DISABLE: Deactivates a trigger macro.

PRIOR : Defines the priority of a trigger (variable).

7.1 TM. TRIGGER MACRO DEFINE

The directive is used to define a trigger macro.

The trigger macro consists of 4 parts: *Name*, *Condition*, *Body* and *Priority*, and is defined in the following way:

```
^TM/Name/Condition/Body/Priority;
```

The same rules apply for *Name* as for macro names in general.

Condition is a logical expression following the same syntax rules as logical expressions in the IF directive. This is explained in section 9.3 on page 59.

For *body* you apply the same rules as for user macro bodies. However, it is important to remember that the expansion of the body MUST lead to a change in one or several parameters in *condition*. If this is not so, the macro will go into an eternal loop. Since a trigger macro is never explicitly called, parameters cannot be included.

The parameter *priority* sets the macro's priority. This parameter is optional, the default value is 0 and authorized values are from 0 to 32768. By setting trigger macro priorities you may decide the order of their expansion in case of a conflict. A conflict may arise if two macros' conditions are fulfilled simultaneously.A condition in a trigger macro is built up by the relations between variables and constants. It is therefore an obvious requirement that the condition contain at least one variable (trigger). A trigger may be either an integer macro or one of the following system macros:

$PN
$CN
$CLINE
$SECLEV

When a trigger macro is defined the system will create an overview of the triggers (variables) included in the expression. Each time one of these triggers is modified, all the trigger macro conditions in which that particular trigger is included will be tested. For each fulfilled condition the corresponding macro will be triggered.

Ex-24: *^TM/MARK/^<^$CLINE; = 5^>/^<-----^BL;^>;*

The trigger macro's condition in the example contains a trigger (variable), namely $CLINE. This means that each time this trigger is modified, ie., each time the line number changes, the trigger macro condition will be checked. In those cases where the condition is fulfilled, the body is expanded. This means that line number 5 on each page will contain only -----

.

When a trigger macro is being expanded, no other trigger macro can interrupt the process until the whole body has been expanded. If the expansion of a trigger leads to the modification of one or several triggers, a situation may arise where several trigger macros are in an expansion queue.

Ex-25: *^TM/MARK/^<^$CLINE; = 5^>/^<-----^BL,1;^>;*
^TM/LINE/^<^$CLINE; = 6^>/^<.....^bl;^>;

In this case we have two trigger macros which both depend upon the trigger $CLINE. The macro MARK is triggered on line 5 and creates 2 lines. This means that it also creates line 6 before the expansion is through. The macro LINE cannot, therefore, obtain access where it is supposed to, ie., on line 6. It is therefore put in a queue and has to wait until the MARK macro has been fully expanded.

The example is simple, but similar situations may lead to a long queue of trigger macros awaiting access.

The priority in this queue is dependent upon the priorities of the triggers involved. The priority may be implicitly set with the PRIOR directive. See section 7.10. If a situation arises where two trigger macros are triggered by the same variable ($CLINE, for instance), the order in which the two are expanded will depend upon the trigger macro priority. This can be specified in the trigger macro definition.

It is important to note that if a trigger macro is to be expanded in a loop when it is first triggered, it will be entered into the queue between each expansion. This therefore means that a trigger macro can be interrupted between two expansions of the body.

It would appear necessary here to provide a complete example:

Ex-26:
```
^TM/LINE/^<^$CLINE; =10^>/^<^PRINT-LINE;^>;
^TM/PAGE/^<(^$PN; = 5) AND (^$FIGSIZE <= 15)^>/
^<^PRINT-FIG;^>;
^PRIOR/CLINE/10;
^PRIOR/PN/5;
```

We assume that the macro PRINT-LINE generates a line with a certain result.

We also assume that the macro PRINT-FIG goes through 15 loops, each loop generating one line.

On each page up to page 5 the macro LINE is triggered on line 10. When you reach page 5 the macro PAGE is triggered, and starts to produce a result. Once line 10 is reached, the macro LINE is triggered. PAGE was triggered by $PN, whereas LINE was triggered by $CLINE. Since $CLINE has the higher priority of the two, the macro LINE will be expanded. Once LINE has been expanded, the PAGE macro may terminate its expansion.

This example showed how trigger priority functions. As mentioned earlier, the trigger macro itself also has a priority. The below example is an illustration:

Ex-27:
```
^P-LINE=10;
^TM/LINE/^<^$CLINE; = 10^>/^<^PRINT-LINE;^>/10;
^TM/FIG-T/^<^$CLINE; = ^$P-LINE;^>/ ^<^PRINT-FIG;^>/5;
```

In this case both trigger macros are triggered by $CLINE when this gets value 10. The trigger priority is therefore the same here. Since LINE has the higher priority, it will be expanded first. When it is finished CLINE will have got value 11, but since FIG-T is already in the queue it will nevertheless be expanded.

If a trigger macro which produces a result attempts to interrupt an implicit line feed caused by a full text line, the result will in most cases be unfavorable. An implicit line feed occurs when you use filling mode 'conditional' or 'filling' and NOTIS-TF decides where to break the lines. In such cases it is therefore necessary top use FM=N, with which the layout of the line can be set in advance in the text editor. If you use trigger macros with $CLINE you thus have to use FM=N.

7.2 MK. MACRO KILL

This directive can be used with trigger macros in the same way as with user macros (see page 29).

7.3 MA. MACRO APPEND

This directive can be used with trigger macros in the same way as with user macros (see page 30).

7.4 MI. MACRO INSERT

This directive can be used with trigger macros in the same way as with user macros (see page 31).

7.5 MR. MACRO REMOVE

This directive can be used with trigger macros in the same way as with user macros (see page 32).

7.6 DM. DUMP MACRO

This directive can be used with trigger macros in the same way as with user macros (see page 35).

7.7 FIX. FIX MACRO

This directive can be used with trigger macros in the same way as with user macros (see page 36).

7.8 DISABLE, DEACTIVATE A TRIGGER MACRO

With this directive you can deactivate a trigger macro. By deactivating a trigger macro you prevent it from being triggered no matter which state the implicated triggers are in.

The syntax for this directive is as follows:

^DISABLE/name/name/.../name;

where *name* is the name of a trigger macro to be deactivated. As you will see from the syntax it is possible to deactivate several trigger macros in one call.

7.9 ENABLE, ACTIVATE A TRIGGER MACRO

This directive is the opposite of DISABLE, ie., it activates a passive trigger macro.

The syntax for this directive is as follows:

^ENABLE/name/name/.../name;

where *name* is the name of the trigger macro to be activated. As you can see from the syntax it is possible to activate several trigger macros in one call.

7.10 PRIOR, SET TRIGGER (VARIABLE) PRIORITY

The syntax for this directive is as follows:

^PRIOR/name/priority;

where *name* is the name of a trigger macro or of a system macro that can be used as a trigger in a trigger macro.

Priority is a numeric value in the area 0 - 32767.

7.11 SINGLE-TRIGGERS

There is one special type of trigger macro, the so-called single-trigger (ST). This is a macro which is expanded in the same way as a trigger macro, but automatically deleted after the first expansion. A single-trigger is never expanded more than once. It has to be defined again after expansion if it is to be used again.

A single-trigger is defined in the following way:

^*ST/name/condition/body/priority;*

where the parameters follow the same rules as for trigger-macros.

8 DETAILED DESCRIPTION OF THE MACRO TYPE TRIGGER STRINGS

Trigger strings are a predefined type of single-triggers.

They have predefined *names*, *conditions* and *priorities*, but the body can be defined by the user.

Contrary to what happens to single-triggers, only the body is deleted after expansion but not the macro itself.

There is a system macro for each trigger string, a *flag* which indicates whether the string is in use or not. The flags are meant to be used for testing with the IF directive (see section 9.3). They have value 1 if the trigger string is active, and value 0 if it is passive.

A trigger string body is defined in the following way:

```
'NAME/body;
```

where *name* is the predefined macro name.

The same rules apply for *body* as in trigger macros.

The following trigger strings are defined in the system.

TP-STR : Title page string

PH-STR : page header string

TL-STR : Trailer string

EOF-STR: End of file string

EOD-STR: End of document string

PAR-STR: Paragraph string

UND-STR: Underline string

It is important to note that the definition of a trigger string body is equal to MA (macro append) for trigger macros. Ie., the new body will be appended to a possible existing body in the string.

8.1 TP-STR. TITLE PAGE STRING

This is a string which is triggered before the first character is written in the out-document. This means that if the trigger string has a defined body, this will be expanded at the time when the system tries to write the first character in the formatted document.

As can be seen from the name, this trigger string is used to make a title page for the document.

8.2 PH-STR. PAGE HEADER STRING

This trigger string is triggered after the top border has been processed, ie., immediately after the page header.

There is a system macro for this trigger string:

```
'$F-PH-STR;
```

which gives value 0 for an empty body and value 1 for a defined body.

An example of use may be the case where you want to move a block of text over to the next page without terminating the current page. See the FIG-CP macro in chapter 10

8.3 TL-STR. TRAILER STRING

This string is triggered in the first line of the bottom border. When you use it, be careful that there is room for the trailer string in the bottom border. The trailer string is written in the last line of the bottom border.

There is a system macro for this trigger string:

```
'$F-TL-STR;
```

which gives value 0 for an empty body and value 1 for a defined body.

A typical example of the use of this trigger-string can be the generation of footnotes. See the FOOTNOTE macro in the macro library.

8.4 EOF-STR, END OF FILE STRING

The trigger string is triggered at the end of the current in-document. Each in-document has its own EOF-STR. When you define a body for EOF-STR, it is therefore valid for the current in-document.

There is a system macro for this trigger string:

```
^$F-EOF-STR;
```

which gives value 0 for an empty body and value 1 for a defined body.

A typical example of the use of this trigger string can be the emptying of a figure-queue at the end of each in-document, as in the help macro *CLEANUP* used by the *FIG-CP* macro.

8.5 EOD-STR, END OF DOCUMENT STRING

This trigger string is triggered at the end of the document, after a possible EOF-STR defined for the main in-document.

There is a system macro for this trigger string:

```
^$F-EOD-STR;
```

which gives value 0 for an empty body and value 1 for a defined body.

An example of the use of this trigger string can be the definition of a reference macro containing the page number for the last page. This can be used for instance when you want to write letters where the page numbering should be carried out according to the formula *This is page x of n*, where *n* is the total number of pages.

The trigger string may also be used for various other types of 'cleaning-up' to be carried out at the end of a document.

8.6 PAR-STR. PARAGRAPH STRING

If this trigger string is defined it is triggered at the start of a new paragraph instead of the usual paragraph-handling directives. This is to say that no paragraph justification is carried out.

There is a system macro for this trigger string:

```
$F-PAR-STR;
```

which gives value 0 for an empty body and value 1 for a defined body.

8.7 UND-STR. UNDERLINE STRING

If this trigger string is defined it is triggered by underline on/off instead of the usual underline handling directives. This is to say that no underline on/off will be carried out.

There is a system macro for this trigger string:

```
`$F-UND-STR;
```

which gives value 0 for an empty body and value 1 for a defined body.

An example of the use of this trigger string can be the redefinition of underlining to be a change in printer font, so that underlined text blocks in the in-document come out as text printed with another font in the out-document.

9 SPECIAL DIRECTIVES FREQUENTLY USED FOR MACRO HANDLING

In this chapter we shall explain the use of some special directives and techniques frequently used for macro handling. We shall also further explain levels and the use of directive quotes.

9.1 THE DIRECTIVES OF AND NF

We shall start by explaining how the NF and OF directives function, since these are frequently used for macro handling (see also NOTIS-TF Reference Manual - Text Formatter, ND-63.007.02, section 3.1.14):

The NF directive leads to storage of certain format parameters:

Left margin	($LM)
Right margin	($RM)
Filling mode	($FM)
Justification mode	($JM)
Left border	($LB)
Right border	($OB)
Page width	($PW)
Text width	($TW)
Horizontal pitch	($HP)
Vertical pitch	($VP)
Underline status	(underline on/off)
Bold text status	($BT)
Font number	($FONT)

After the NF directive you may modify this set of formats. When you later input a call for the OF directive, the previous set of formats stored with the NF directive will be called up and used as the current set again. The set of modified formats will disappear.

It is possible to nest NF and OF calls. You must, of course, not have more OF than you have NF calls. The opposite is of no consequence, however.

9.2 THE DIRECTIVE AR

The AR directive is used to carry out arithmetic calculations on figures in numerical representation. This means that Roman numerals or alphabetically represented values may not be used in these calculations. The directive consists of two parts, name and expression, and is called in the following way:

```
^AR/expression;
```

Expression can be a combination of numeral values and the following operators:

\+ Addition

\- Subtraction

/ Division

* Multiplication

() Parentheses for order of priority

The general rule is that the expression is executed from the innermost parenthises level and outwards. For operators on the same level, multiplication and division have the same priority. This priority is higher than for addition and subtraction, which also have equal priority.

If one or several right parentheses are omitted in the expression, these will be added at the end so that the calculation can be carried out. The result will then of course be wrong, but you will receive a message on the screen telling you this during formatting. The multiplication sign before an expression enclosed in parenthesis may be omitted.

Ex-28: *^AR/19 + 3(^$WAGES; - 4);*

Obviously the arithmetic expression is not calculated until the whole AR call has been read. In this example, this leads to the expression to be calculated looking like this:

19 + 3(n - 4)

where **n** *is the value of the integer macro* **WAGES**. *The expression will be calculated in the following way:*

The value 4 is subtracted from **n**, *which is in turn multiplied by 3. The result is added to 19.*

The result of an AR directive is written where the directive is input, thus following the same rules as user macro calls.

For further information, see the example of the FIG-CP macro on page 81, where the AR directive has been used.

9.3 THE DIRECTIVE IF

The IF directive is used to test logical expressions. The directive consists of four parts: *name*, *condition*, the *THEN (true) body* and the *ELSE (false) body*, and is called in the following way:

```
'IF/condition/THEN/ELSE;
```

where ELSE is optional.

Condition is a logic expression, and the following operators are authorized:

AND : Logical relation

OR : Logical relation

NOT : Negation

MISD: Macro IS defined

MIND: Macro IS NOT defined

ODD : Test for odd numbers

EVEN: Test for even numbers

' : Apostrophe for string comparison

() : Parentheses for isolation of parts of logic expression

= : Equal to

<> : Different from

>< : Different from

< : Smaller than

> : Greater than

<= : Smaller than or equal to

=< : Equal to or smaller than

>= : Greater than or equal to

=> : Equal to or greater than

The same rules apply for *THEN* and *ELSE* as for user macro bodies.

The condition may be either *TRUE* or *FALSE*. If it is true, the THEN body will be expanded. If it is false, the ELSE body will be expanded.

Ex-29: *^IF/odd(^$pn;)/odd numbers/even numbers;*

In this call we are testing whether the current page has an odd or an even number. If it has an odd number, the text **odd numbers** *will be written out. If the opposite is true, the text* **even numbers** *will be written out.*

On the present page the call will result in:odd numbers

In the next example we assume that we have already defined a a macro called *DAY*, which gives today's name as a result.

Ex-30:
```
^IF/(^$DD; = 20) AND
    ('^DAY;' <> 'Saturday') AND
    ('^DAY;' <> 'Sunday')/Payday!/No money to be had;
```

In this call we are testing whether it is payday. The criteria is that it must be the 20th of the month, and that it must be neither Saturday nor Sunday. When this example was written, it resulted in:

Payday!

For *IF* as for all other directive/macro calls, the rule is that the whole call is read before it is executed. In the present case this means that the condition will not be tested until the whole call has been read. This can be important when strings (bits of text) are being compared.

If, for instance, we have a macro, *NAME*, containing the text *Allison's* which is to be used in a string comparison, this has to be carried out in the following way:

Ex-31: *'IF/'<''NAME;' = 'Lloyds''>/.....*

After the IF has been read, the condition will look like this:

''NAME;' = 'Lloyds'

This means that **'NAME;** *is expanded by analysis of the condition. As a result, the apostrophe contained in the macro* **NAME** *appears on a higher level than the apostrophes enclosing* **NAME**, *and no confusion arises*

If the expression had been written without quotes, like this:

'IF/''NAME;' = 'Lloyds'/....

the condition would have looked like this after the IF call was read:

'Allison's' = 'Lloyds'

The result here is, therefore, that the apostrophe inside **NAME** *ends up on the same level as the apostrophes enclosing* **NAME**, *and cannot be distinguished from them. This again leads to a misinterpretation of the expression, and results in an error message.*

9.4 USE OF MACRO LIBRARIES WITH THE DIRECTIVE LIB-CONT

Any program will have a limited storage capacity. For NOTIS-TF, this means that there is a limit to the number of macros you may define. This limits depends more on the sum of the macros' sizes than on the actual number of macros. To get around this problem it is possible to store macro definitions in library files. The directive *LIB-CONT* is used for this purpose.

The LIB-CONT directive is called in the following way:

'LIB-CONT/Name-1/Name-2/Name-3/..../Name-n;

The whole call has to be written on the same line.

When a macro library is being included it is initially read up to the last LIB-CONT directive. The LIB-CONT directives do not necessarily have to be at the beginning of the library file, but all LIB-CONT directives must be consecutive.

A macro library will therefore normally be built up like this:

Definition of small, global help macros, as well as integer macros, etc.

Then the LIB-CONT for all larger macros.

Finally, the definition of these larger macros.

When such a library is being included, all the definitions are read up to the first LIB-CONT. Then all the LIB-CONT calls are read, and the reading stops there temporarily.

When LIB-CONT is used, the macro names given as parameters to the directive are defined as user macros. However, the body itself (the part that requires the most space) remains undefined.

When one of the macros defined in LIB-CONT is subsequently called, NOTIS-TF will continue to read the library file until it encounters the definition of the macro in question. This therefore means that you may have access to a large number of macros, but only those that are being used occupy space in the computer's memory.

A macro library is included in the usual way, with the directive *^IN/Name/L;*. The L at the end of the directive tells NOTIS-TF that the file is to be treated as a library.

If you want to add new macro definitions to the standard macro library supplied with NOTIS-TF, it is advisable to store them in a separate library which you then include in the standard library. The inclusion has to take place before the first LIB-CONT. If you do it in this way, it is easier for you to upgrade new versions of the standard library by simply including your own macro library or libraries.

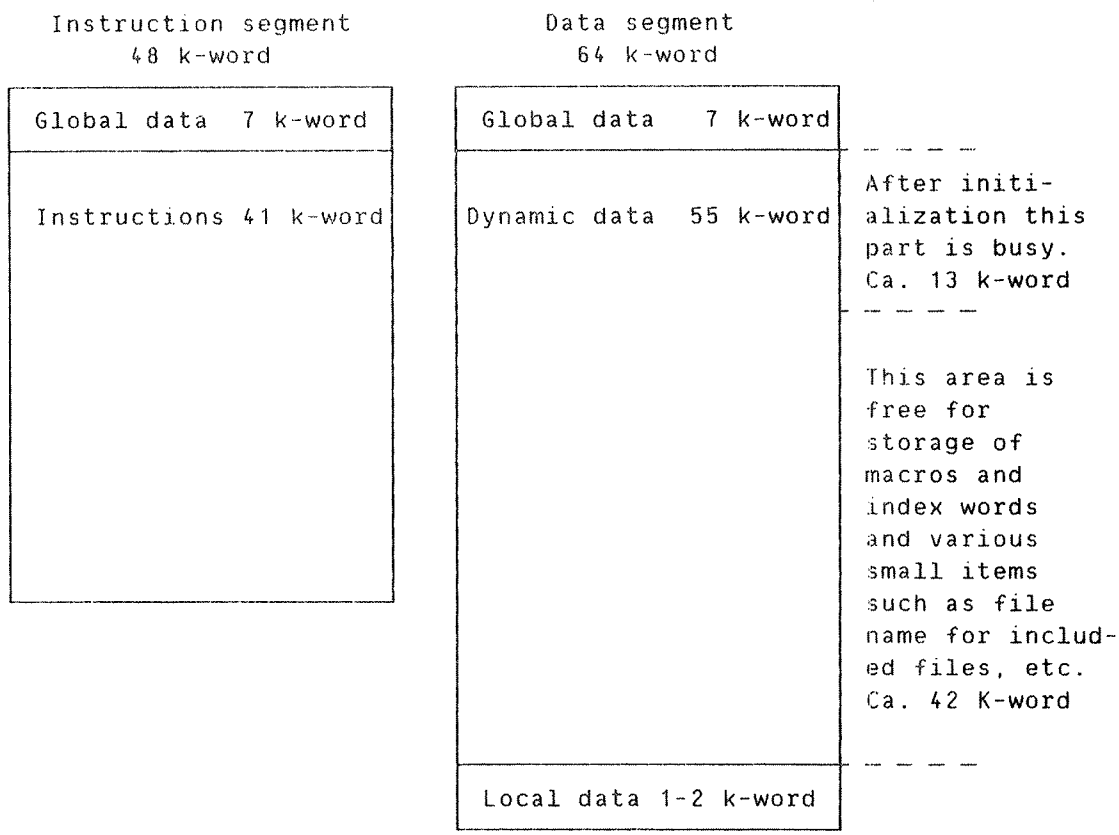
In practice, the maximum number of libraries you will be able to include is about 10, depending to some extent on the depth of inclusion levels in your text documents.

9.5 NOTIS-TF - RESOURCE AND SPACE REQUIREMENTS IN THE COMPUTER

This section provides a more detailed explanation of the way NOTIS-TF functions internally, and some knowledge of electronic data processing is needed to benefit from it. The section is therefore of interest only to some of you, and may be skipped by the others.

NOTIS-TF is a typical batch-oriented application. This means that it carries out its functions with little or no interactive influence from the user. It also means that it does not need to wait for anything while it works, and it is therefore normal that it utilizes all the available resources in the computer. This may be a disadvantage for interactive users competing with NOTIS-TF for computer resources.

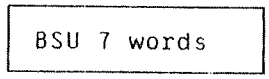
NOTIS-TF is executed as a 'two-bank' system in ND-10/ND-100 and ND-SATELLITE. This means that it uses one segment for instructions and one segment for data. The division can be illustrated as follows (all values used here are approximate):



As can be seen in the illustration, the remaining available space in the dynamic data field is used to store macros and index words. The occupied part of this dynamic data field is used to store all directives, as well as all macro heads defined in the library. This means that if the library is extended, or more libraries included, this part will increase.

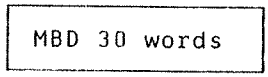
To give you an idea of the amount of space occupied by the various items we shall explain to you the most current data elements.

The basic data element is called Basic-String-Unit (BSU). It is a unit used for storage of all types of character strings, such as macro names, macro bodies, index words, document names, etc.

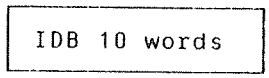


1 BSU can store up to 8 characters.

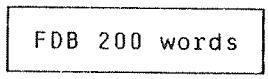
As explained earlier, a macro consists of a head, a name and a body. This head is also a data element, Macro-Descriptor-Block (MBD).



An index word consists of a head plus the word itself, Index-Descriptor-Block (IDB).



Any file handled by NOTIS-TF is given a file head, File-Descriptor-Block (FDB). The system will not allocate more FDB's than there are concurrently opened files. This means that if several files are included on the same level, they will use the same FDB.



Based on this explanation we shall give a few examples of how to calculate the space requirements for the various functions.

If you define a macro as follows:

```
`MD/TITLES/Here are two titles;
```

it will have the following space requirement:

```
1 MDB                                = 30 words
Name    =  6 char.):  1 BSU    =  7 words
Body    = 17 char.):  3 BSU    = 21 words
-------------------------------------------
Total space requirement        = 58 words
===========================================
```

If you define an index word as follows:

```
`XA/Next calendar-year;
```

it will have the following space requirement:

1 IDB	= 10 words
Name = 18 char.): 3 BSU	= 21 words
Total space requirement	= 31 words

As mentioned earlier, all these values are approximate and must therefore be given a certain margin.

In this context we shall take a further look at the effect of the LIB-CONT directive used in macro libraries. What happens is that NOTIS-TF reads the file up to and including the last LIB-CONT directive. All macros contained in these LIB-CONTs will then be defined, but in the memory they will only occupy 1 MDB plus the name. Now, when they are used, the body itself will be read from the file. This is why you can define many large macros, but only be able to use a few of them at the same time.

If a macro is deleted, the space it occupied will be made available for a new macro definition.

10 EXAMPLES

In this chapter we shall attempt, through examples, to illustrate the practical use of macros.

The complexity of the macros will increase throughout the chapter.

Most of the examples are based on the standard NOTIS-TF macro library.

10.1 THE MACRO BOLD

The function of this user macro is to cause text to be printed in bold print.

A call to this macro may look as follows:

...something that is ^BOLD/important;.

The result will be:

...something that is ***important*****.**

Definition: *^MD/BOLD/^<^BT=+;^1;^BT=-;^>;*

The first call in the body is *[^BT=+;]*, which turns bold text mode on.

Then a parameter is included. This is the text which is to be printed in bold print.

Finally, bold text mode is turned off again with the call *[^BT=-;]*.

10.2 THE MACRO SHOW

This macro has more or less the same function as BOLD, but in addition to to being printed in bold, the text is also centered on a new line.

A call to this macro may look as follows:

...this causes the amount of ^SHOW/US$ 275; to be deducted...

which results in

...this causes the amount of

US$ 275

to be deducted...

Definition: *^MD/^<^BL=1;^NF;^BT=+;^JM=C;^1, ;^BL=1;^OF;^>;*

The first call, *[^BL=1;]*, gives a blank line.

The call *[^NF;]* leads to storage of the format parameters.

The call *[BT=+;]* activates bold text mode.

The call *[JM=C;]* sets justification mode centered. This causes all subsequent text to be centered.

Then a parameter is included. The default value of this parameter is a blank. This is not strictly necessary.

After the parameter you find the call *[BL=1;]*, which causes a blank line to be added after the centered parameter.

Finally, there is the call *[^OF;]*. The call causes the format parameters last stored to be fetched out again. In the present example this means that the bold text and justification modes will be set back to the values used before the macro was expanded.

10.3 THE EXAMPLE MACRO IN THIS MANUAL

In this manual we have used a combination of integer macros and other macros types to automatically number the examples.

We have defined the following macros for this purpose:

```
1) ^IM/EX-NO/0;
2) ^MD/START-EX/^<^EX-NO=+1;^NF;^LM=+10;^FM=N;
   ^IP=Ex-^$EX-NO;: ;^>;
3) ^MD/END-EX/^<^OF;^>;
```

We started by defining an integer macro called *EX-NO*, with value 0.

The second macro definition was a user macro called *START-EX*. The effect of this macro is as follows: The example number is first incremented by 1: *[^EX-NO = +1;]*. Then all the format parameters are stored: *[^NF;]*. In the next step the left margin is increased by 10 character positions and filling mode set to no-fill: *[^LM=+10;^FM=N;]*. Afterwards we start an inverted paragraph with the text *Ex-n:*, where *n* is the current example number: *[^IP=Ex-^$EX-NO;: ;]*.

Finally we defined the macro *END-EX*, used to terminate an example. This macro fetches the format parameters previously stored: *[^OF;]*.

An example is thus entered like this:

```
Ex-32: ^START-EX;This is an example....

       ..............
       ..............

       ^END-EX;
```

10.4 THE MACRO BM

The macro is used to set both margins (LM and RM) simultaneously.

Definition: *^MD/BM/^<^LM=^1,0;;^RM=1,0;;^>;*

As you can see, both margins are given the same value. The default value is 0, which means that a call to BM without a parameter sets both margins at 0.

It is important to remember that the right margin starts at the value of the right border (OB).

10.5 THE MACRO CE

The macro is used to center a text on one or on several lines of its own, depending on the amount of text to be centered.

A call to this macro may look as follows:

^CE/This text will be centered on a line;

resulting in:

This text will be centered on a line

Definition: *^MD/CE/^<^NF;^BL;^JM=C;^1;^BL;^OF;^>;*

The first call, *[^NF;]*, causes the format parameters to be stored. They can thus be retrieved once the macro has been expanded.

The next call, *[^BL;]* ensures termination of a non-terminated text line. The call has no effect if the preceding text line is terminated.

The next call is *[^JM=C;]*, which sets justification mode centered. As a result, all subsequent text will be centered.

Then a parameter is included. This will be the text to be centered.

The text line resulting from the included parameter is terminated with a *[^BL;]* call.

And finally, all parameters which may have been modified are reset to their original values with the *[^OF;]* call. In this particular case only the justification mode is involved.

10.6 THE MACRO CP

This macro causes a conditional switch to a new page, ie., if the number of lines left on the page is inferior to a specified number of lines, the system will start a new page. If not, it will terminate the current line of text.

Definition: *^MD/CP/^<^IF/^AR=^$PL;-^$CLINE;-^$BB;+1;< ^1;*
/^<^PG;^>/^<^BL;^>;^>;

In the first line we set a condition by calling the IF directive. The condition consists of testing whether the number of the remaining lines on the page is inferior to the number specified.

The number of remaining lines is calculated with the AR directive (see page 57), like this:

```
  Page length            ($PL    )
- Current line number    ($CLINE)
- Bottom border          ($BB    )
+ 1
-----------------------------------------
= The number of remaining lines on the page
=========================================
```

This number is now tested against the number included as parameter 1.

If the condition is fulfilled, the first body in the IF directive is executed; in this case *[^PG;]*, which causes the switch to a new page.

If the condition is not fulfilled, the second body in the IF directive is executed; in this case *[^BL;]*, which causes the current line of text to be terminated. The call has no effect if the current line is terminated.

10.7 THE MACRO SPREAD

The macro writes three text strings on the same line, justifying them left, center and right, respectively. The line is followed by a specified number of blank lines.

A call to this macro may look as follows:

```
^SPREAD/SMITH/BROWN/JOHNSON/2;
```

resulting in:

SMITH *BROWN* *JOHNSON*

Definition:
```
^MD/SPREAD/^<^BL;^NF;^FM=F;^JM=L;^1;^BL=-1;
^JM=C;^2;^BL=-1;^JM=R;
^IF/'^3,*-*;'<>'*-*'/^3;/^<^BL=1;^>;
^BL=^4,0;;^OF;^>;
```

The call *[^BL;]* leads to the termination of the current text line. If there is none, the call is without effect.

Then the format parameters are stored with the *[^NF;]* call.

Filling mode filling is set with the *[^FM=F;]* call, to cause possible line feeds in parameters or macro body to be ignored.

Justify mode is set to left justification with the *[^JM=L;]* call.

Parameter 1 is then included, and justified left on the line.

The text line is terminated with the *[^BL=-1;]* call. This call causes everything that is found on the current line to be written out, and the same line to be started again. This means that no line feed is carried out.

Justification mode is now set to centered with the *[^JM=C;]* call.

Parameter 2 is included at this point. The text in this second parameter will be centered on the same line where parameter 1 was left justified.

Parameter 2 is terminated in the same way as parameter 1, with a *[^BL=-1;]* call.

Justification mode right is now set with the *[^JM=R;]* call. In this way parameter 3 is justified right.

It is not a necessity to have a third parameter in this call. A test is therefore carried out now, to determine whether a third parameter has been specified in the call. You will find that this type of testing is carried out in several of the macros in the macro library. The test is performed with the IF directive, as follows:

> Parameter 3 is included in single quotes, to indicate that it is a comparison of strings.
>
> The parameter is included with default value *-*, meaning that if it is not specified in the call, the first string in the comparison will be *-*. Also, note that the inclusion of parameter 3 is NOT done between quotes. If it had been, it would have meant that parameter 3 to the IF call should be included.
>
> The comparison is carried out with the operator <> (different from).
>
> The other string in the comparison is *-*.
>
> The result is, that if parameter 3 is included and has a value different from *-*, the result of the comparison will be TRUE. In the opposite case it will become a comparison of two identical strings, and the result will be FALSE.

If the result of the comparison is TRUE, the first body in the IF directive is executed. This means that parameter 3 is included. Note here that the inclusion of parameter 3 is NOT in quotes. If it had been, the significance would have been that parameter 3 in the IF directive should be included, and this is not what we wanted.

If the result of the comparison is FALSE, the second body in the IF directive is carried out, ie., a blank line is inserted.

The number of blank lines specified in parameter 4 is then inserted. Default is 0.

Finally, the format parameters are reset to their original values with the *[^OF;]* call.

10.8 THE MACRO FIG-CP

The macro makes room for a figure, inserts the figure caption and numbers it with the current number. If there is insufficient space for the figure on the current page, it is moved over to the next. However, the text continues on the current page. The figure is in this case moved from the text, but all references to the figure number is maintained. If the figure is bigger than maximum page size an error message appears on the screen during formatting, and an error message also appears in the text at the location where you tried to place the figure.

A call to this macro may look like this:

```
^FIG-CP/7/We could have inserted a figure here;
```

resulting in:

Fig. 1. We could have inserted a figure here

Definition:

```
^MD^FIG-CP^
^<^IM/FIG-ROOM/^1,20;;^MD^FIG-TEXT^^PM,2;;^FN=+1;^BL;
  ^IF|^AR=^$FIG-ROOM;+4;>^AR=^$PL;-^$TB;-^$BB;;|
  ^<^MS^** Error in use of macro FIG-CP, fig. ^$FN;, -too many lines **;
     ^BL=2;^NF;^BM=0;
    ^JM=c;_Error in use of macro FIG-CP, fig. ^$FN;, -too many lines_
    ^BL=2;^OF;
  ^>|
  ^<^IF|(^AR=^$FIG-ROOM;+4;>^AR=^$PL;-^$CLINE;+1-^$BB;;)OR(^$FIG-PN;=^$PN;)|
    ^<^PH-STR=^<^FIG-CP!^>^$FIG-ROOM;!^FIG-TEXT;^<;^>;
      ^FIG-WAIT=+1;^FIG-PN=^$PN;;^CLEANUP;^>|
      ^<^IF|^$FIG-WAIT;>0|^<^FN=^$FN;-^$FIG-WAIT;;^FIG-WAIT=0;^>;
        ^BL=^$FIG-ROOM;+1;
        ^NF;^BM=0; ^JM=C;_Fig. ^$FN;. ^FIG-TEXT;_^BL=2;^OF;
      ^>;
    ^>;
  ^MK/FIG-TEXT;^MK/FIG-ROOM;
^>;
```

The macro may look very complicated and frightening at first, but by splitting it up into sections we make it easier for ourselves. Let us split it up into 5 logical sections, therefore:

1

```
^MD`FIG-CP`
^<^IM/FIG-ROOM/^1,20;;^MD`FIG-TEXT``^PM,2;;^FN=+1;^BL;
```

This section carries out all necessary initializations, such as defining help macros, etc.

2

```
  ^IF|^AR=^$FIG-ROOM;+4;>^AR=^$PL;-^$TB;-^$BB;;|
  ^<^MS`** Error in use of macro FIG-CP, fig. ^$FN;, -too many lines **;
    ^BL=2;^NF;^BM=0;
    ^JM=c;_Error in use of macro FIG-CP, fig. ^$FN;, -too many lines_
    ^BL=2;^OF;
  ^>|
```

This section takes care of testing, and error handling too if the figure is bigger than the maximum authorized size.

3

```
^<^IF|(^AR=^$FIG-ROOM;+4;>^AR=^$PL;-^$CLINE;+1-^$BB;;)OR(^$FIG-PN;=^$PN;)|
  ^<^PH-STR=^<^FIG-CP!^>^$FIG-ROOM;!^FIG-TEXT;^<;^>;
    ^FIG-WAIT=+1;^FIG-PN=^$PN;;^CLEANUP;
  ^>|
```

This section tests whether it is possible to insert the figure on the current page, if not it puts it in a queue for the next page.

4

```
  ^<^IF|^$FIG-WAIT;>0|^<^FN=^$FN;-^$FIG-WAIT;;^FIG-WAIT=0;^>;
    ^BL=^$FIG-ROOM;+1;
    ^NF;^BM=0; ^JM=C;_Fig. ^$FN;. ^FIG-TEXT;_^BL=2;^OF;
  ^>;
```

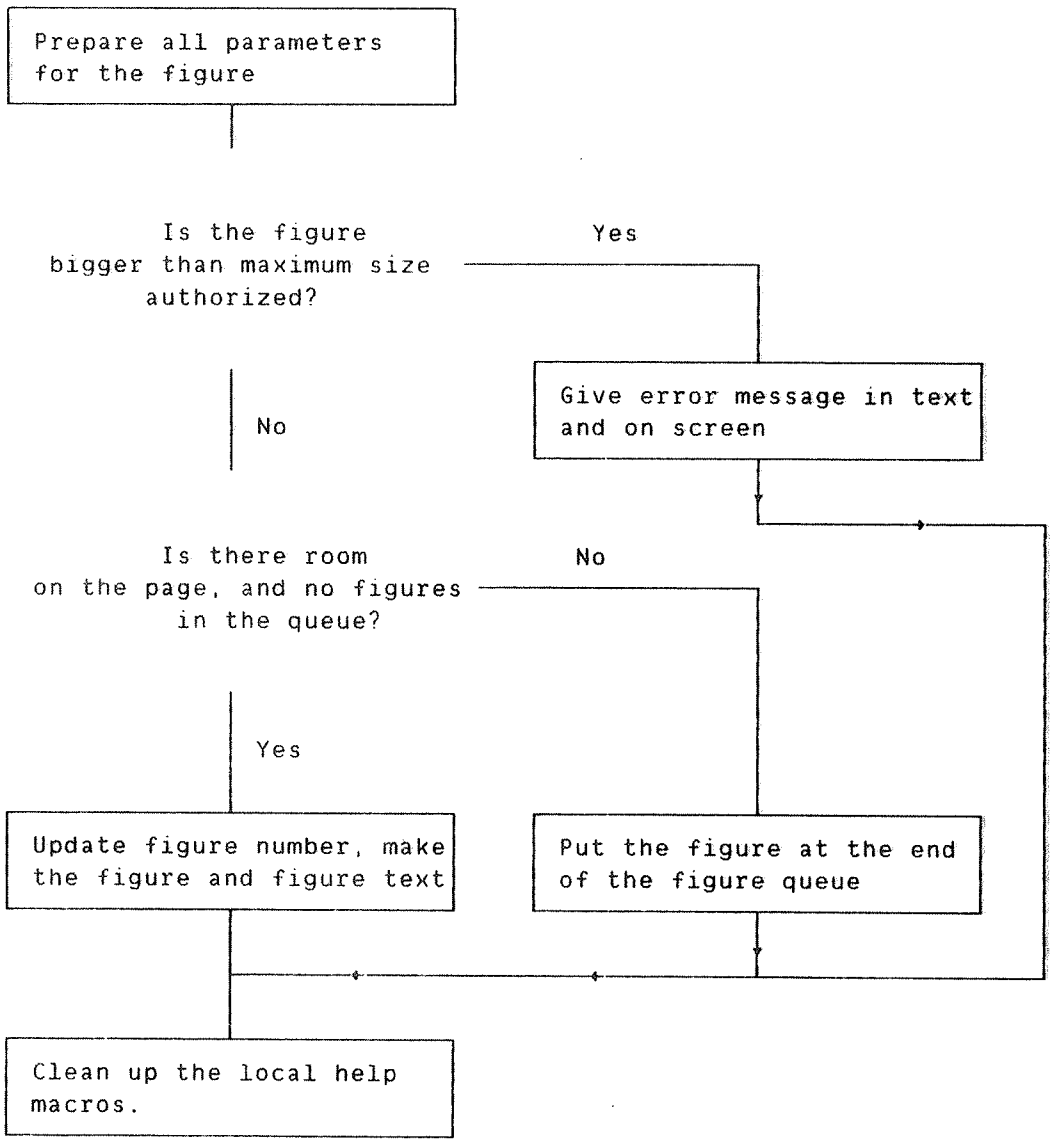
This section carries out the figure handling itself, ie., reserves the space required.

5

```
  ^>;
  ^MK/FIG-TEXT;^MK/FIG-ROOM;
^>;
```

This section does the necessary cleaning up, such as removing local help macros.

A flowchart may contribute to make this macro even easier to understand:



Now that the logical function of each section has been described, we shall describe the details in each individual section. Before doing this, however, we have to describe the relevant global help macros:

FN : This is an integer macro containing the current figure number.

FIG-WAIT: This is an integer macro containing the number of figures in the queue for the next page.

FIG-PN : This is an integer macro containing the page number of the page where you attempted to make the figure lying last in the queue.

CLEANUP : This is a macro that defines the operations to be carried out in order to empty the figure queue after the last page. If the figure queue is empty, no action will be taken.

DETAILED DESCRIPTION OF SECTION 1 IN THE FIG-CP MACRO

We start by defining the integer macro FIG-ROOM to contain the value of parameter-1 in the FIG-CP call, the number of lines to be reserved for the figure. The default value is 20. *[^IM=FIG-ROOM=^1,20;;]*.

Thereafter we define the macro FIG-TEXT to contain parameter-2 in the FIG-CP call, the caption: *[^MD^FIG-TEXT^^PM,2;;]*. We have used the PM directive here because we want to be able to include a directive in parameter-2, such as a directive for left justification of the text, for instance.

Then the current figure number incrementation is ensured with *[^FN=+1;]*, causing the figure to be numbered one number higher than the previous one.

Finally the current text line is broken with *[^BL;]*. If the current text line is empty, this call is without effect.

DETAILED DESCRIPTION OF SECTION 2 IN THE FIG-CP MACRO

Here we use *[^IF]* to test whether the figure plus 4 lines *[^FIG-ROOM;+4]* is bigger than the number of lines available on a page of text *[^AR.^$PL;-^$TB;-^$BB;;]*. The sum of available lines is therefore calculated as page length minus top border minus bottom border. As mentioned earlier, this sum is tested against the figure size plus 4 lines. Those 4 lines are needed to make room for the caption plus one blank line over and under it.

This IF directive has a THEN part and an ELSE part. The THEN part, which is expanded if the test gives a TRUE value, contains the remainder of section-2. The ELSE part, which is expanded if the test gives a FALSE value, contains sections 3 and 4. This means that the remainder of section 3 plus section 5 are expanded if the figure is too big for the page (TRUE value). In the opposite case section 3, possibly section 4, and section 5 are expanded.

The expansion of the remainder of section 2 causes a message to be written on the screen *[^MS]*, containing the figure number [5[^$FN;]. The same message is also inserted in the text at the location where an attempt was made to reserve space for the figure. The message in the text starts with 2 blank lines *[^BL=2;]*. The format parameters are stored *[^NF;]*. Margins are set to 0 *[^BM=0;]*. Justify mode is set centered *[^JM=C;]* The message containing the figure number *[^$FN;]* is now written, underlined and centered on a line. The message is terminated with 2 blank lines *[^BL=2;]*. Finally, the format parameters are set back to their original values *[^OF;]*, in this case the margins.

DETAILED DESCRIPTION OF SECTION 3 IN THE FIG-CP MACRO

This section also starts with a test *[^IF]* to check whether the figure plus 4 lines *[^$FIG-ROOM;+4]* is bigger than the number of lines left on the current page *[^AR.^$PL;-^$CLINE;+1-^$BB;;]*. The sum of remaining lines is therefore calculated as the sum of page length minus current line plus 1 minus bottom border. As mentioned earlier, this sum is tested against figure size plus 4 lines. These 4 lines are needed for the caption plus a blank line over and under it. The IF directive in fact contains 2 tests with the logic relation OR. This means that if one of the tests gives the value TRUE, the total result becomes TRUE. The other tests consists of checking whether attempts have been made earlier to reserve space for figures on the current page, and if such figures are in the queue for the next page *[^$FIG-PN;=^$PN;]*. If this is the case, the current figure also has to be put in the queue for the next page, whether there is room for it or not on the current page. Otherwise the order in which the figures are to be printed might be modified.

The IF directive has a THEN part and an ELSE part. The THEN part, which is expanded if the test gives the value TRUE, contains the remainder of section 3. The ELSE part, which is expanded if the test gives the value FALSE, contains section 4. This means that if there is insufficient space for the figure on the current page (TRUE value), the rest of section 3 and section 5 are expanded. In the opposite case sections 4 and 5 are expanded.

The expansion of the remainder of section 3 causes this FIG-CP call to be entered into the page header string *[^PH-STR]*, to be expanded again at the top of the next page. If there are already FIG-CP calls in the page header string, the new call is put in a queue after the others. Then the count for the number of figures in the queue is increased by one *[^FIG-WAIT=+1;]*, and the page number for the last figure in the queue for the next page is set to the current page number *[^FIG-PN=^$PN;;]*. At the end of this section the macro CLEANUP is called to activate, if necessary, the cleaning up of the figure queue after the last page of the document. A call to this macro only has effect the first time, and repetitive calls are therefore meaningless.

DETAILED DESCRIPTION OF SECTION 4 IN THE FIG-CP MACRO

This is the section that reserves space for the figure and prints the caption.

It starts by testing *[^IF]* whether there are figures in the queue *[^$FIG-WAIT 0]*. If this is the case, the current figure must also be in the queue. The figure number must therefore be adjusted to the number of figures waiting *[^FN=^$FN;-^$FIG-WAIT;;]*. In addition, the count for the number of figures in the queue must be set back to 0 *[^FIG-WAIT=0;]*.

Space is now reserved for the figure, plus a blank line over the caption *[^BL=^$FIG-ROOM;+1;]*. The format parameters are stored *[^NF;]*. The margins are set to 0 *[^BM=0;]* to center the caption on the line. Justification mode is set to centered *[^JM=C;]*. The caption containing the figure number is printed and underlined. It is followed by two blank lines *[^BL=2;]*. Finally the format parameters are reset to their original values *[^OF;]*, in this case the margins.

DETAILED DESCRIPTION OF SECTION 5 IN THE FIG-CP MACRO

This section carries out the necessary cleaning up, consisting of the deletion of the macros FIG-TEXT *[^MK/FIG-TEXT;]* og FIG-ROOM *[^MK/FIG-ROOM;]*.

10.9 THE MACRO SIP

This macro is used to start a sequence of inverted paragraphs. Each individual paragraph starts with the macro NP, defined within SIP. The sequence must be terminated with EIP, to be described at the end of this section.

The SIP macro can be called with up to 5 parameters, with the following significance:

1)
This parameter describes how paragraphs created with the NP macro are to be marked. The options are:

N: Consecutive numbering with numeric values.

R: Consecutive numbering with upper case Roman numerals.

r: Consecutive numbering with lower case Roman numerals.

A: Consecutive numbering with upper case characters.

a: Consecutive numbering with lower case characters.

If this parameter is omitted, paragraphs created with the NP macro will be marked with a hyphen '-' in the left margin.

2)
This parameter indicates the width of the left margin. If the parameter is omitted, the default value of 9 is selected.

3)
This parameter indicates how text/margin notations should be justified. The options are the same as for the JM directive. If the parameter is omitted, right justification is selected.

4)
This parameter indicates how the text in the paragraphs should be justified. If the parameter is omitted, the current value of the justifiation mode is selected.

The example below illustrates some of the possible uses of the SIP, NP and EIP macros.

The macro/text sequence below:

```
^SIP/N/3;
^NP;This is the first paragraph with numbers.
^NP;This is the second paragraph with numbers.
^SIP/A/3;
^NP;This is the first paragraph numbered with upper case characters.
^SIP/r/5;
^NP;This is the first paragraph numbered with lower case Roman
numerals.
^NP;This is the second paragraph numbered with lower case Roman
numerals.
^EIP;
^NP;This is the second paragraph numbered with upper case characters.
^EIP;
^NP;This is the third paragraph with numbers
^EIP;
```

results in:

1)
   This is the first paragraph with numbers.

2)
   This is the second paragraph with numbers.

   A) This is the first paragraph numbered with upper case characters.

        i) This is the first paragraph numbered with lower case Roman numerals.

       ii) This is the second paragraph numbered with lower case Roman numerals.

   B) This is the second paragraph numbered with upper case characters.

3)
   This is the third paragraph with numbers.

Definition:

```
^MD/SIP/
^<^NF;^FM=C;
  ^IM=IPNO=1;          ^MD=HOW=^1,*-*;;  ^IM=MARG=^2,9;;    ^LM=+^$MARG;;
  ^IM=IPS=^3,^$PS;;;  ^MD=IJM=^4,R;;    ^MD=JUST=^5,*-*;;  ^BL;
  ^MD/IP/
  ^<^BL=^$IPS;;^CM/MD-SAVP;^MA/MD-SAVP/^PM=1;;
    ^IF|^$LM;<2|
    ^<^MS/** Error in use of macro IP,^MD-SAVP;, LM has to be>= 2. **;^>|
    ^<^CP=^$PF;;^NF;^JM=^IJM;;^IM-SAVP=^$LM;;
      ^IF|''^IJM;'='R'|
      ^<^LM=0;^RM=^$TW;-^$IM-SAVP;;^>|
      ^<^IF|''^IJM;'='L'|
        ^<^LM=-^$MARG;;^RM=0;^>|
        ^<^IF|('^IJM;'='C')OR('^IJM;'='S')|
          ^<^LM=-^$MARG;;^RM=^$TW;-^$IM-SAVP;;^>;
        ^>;
      ^>;
      ^MD-SAVP; ^IF|(^$RM;=0)AND(^$CPOS;>^$IM-SAVP;)|^<^BL;^>|^<^BL=-1;^>;
      ^OF;
    ^>;
  ^>;
  ^IF|''^HOW;'='*-*'|
  ^<^MD/NP/^<^IP=- ;^>;^>|
  ^<^MD/NP/^<^IP=^$IPNO,^HOW;;) ;^IPNO=+1;^>;^>;
  ^IF|''^JUST;'<>'*-*'|^<^JM=^JUST;;^>;
^>;
```

This macro can also be divided into logical sections to make it easier to understand. We have divided it into 4 main sections, with section 2 presented with 5 sub-sections.

1

```
^MD/SIP/
^<^NF;^FM=C;
  ^IM=IPNO=1;          ^MD=HOW=^1,*-*;;  ^IM=MARG=^2,9;;    ^LM=+^$MARG;;
  ^IM=IPS=^3,^$PS;;;  ^MD=IJM=^4,R;;    ^MD=JUST=^5,*-*;;  ^BL;
```

This section carries out the necessary initializations, such as defining help macros, etc.

2.1

```
  ^MD/IP/
  ^<^BL=^$IPS;;^CM/MD-SAVP;^MA/MD-SAVP/^PM=1;;
```

The whole section 2 consists of a defintion of the IP macro, the basis for inverted paragraphs. Section 2.1 carries out the necessary initi-alizations, such as storing parameters, etc.

2.2

```
    ^IF|^$LM;<2|
    ^<^MS/** Error in use of macro IP,^MD-SAVP;, LM has to be>= 2. **;^>|
```

This section tests whether the left margin is wide enough for an inverted paragraph, and provides an error message when needed.

2.3

```
    ^<^CP=^$PF;;^NF;^JM=^IJM;;^IM-SAVP=^$LM;;
```

This section starts the paragraph itself.

2.4

```
      ^IF|''^IJM;'='R'|
      ^<^LM=0;^RM=^$TW;-^$IM-SAVP;;^>|
      ^<^IF|''^IJM;'='L'|
        ^<^LM=-^$MARG;;^RM=0;^>|
        ^<^IF|('^IJM;'='C')OR('^IJM;'='S')|
          ^<^LM=-^$MARG;;^RM=^$TW;-^$IM-SAVP;;^>;
        ^>;
      ^>;
```

This section carries out testing and adjustment of margins for the paragraph.

2.5

```
      ^MD-SAVP; ^IF|(^$RM;=0)AND(^$CPOS;>^$IM-SAVP;)|^<^BL;^>|^<^BL=-1;^>;
      ^OF;
   ^>;
^>;
```

This section creates the inverted paragraph, and terminates the IP macro.

3

```
^IF|''^HOW;'='*-*'|
^<^MD/NP/^<^IP=- ;^>;^>|
^<^MD/NP/^<^IP=^$IPNO,^HOW;;) ;^IPNO=+1;^>;^>;
```

This section defines the NP macro from parameters specified for the SIP call. The NP macro is an alternative to the IP macro in that it provides a standard text in the margin.

4

```
  ^IF|''^JUST;'<>'*-*'|^<^JM=^JUST;;^>;
^>;
```

This section tests and sets justification mode for the text part of the inverted paragraphs to follow, and terminates the SIP macro.

Before describing each section separately we shall describe the individual, global help macros involved:

IM-SAVP :

This is an integer macro used in the IP macro to save the left margin for the paragraph, in order to be able to use it in calculations and tests.

MD-SAVP :

This is also a macro used in the IP macro. It saves the text parameter to be entered into the left margin.

DETAILED DESCRIPTION OF SECTION 1 IN THE SIP MACRO

This section first ensures storage of all format parameters *[ˆNF;]*. Then filling mode is set to conditional *[ˆFM=C;]*.

Thereafter an integer macro is defined *[ˆIM=IPNO=1;]* with a value of 1. This macro is used for consecutive numbering of the inverted paragraphs created with the NP directive.

We now define a macro to save parameter 1 in the SIP call *[ˆMD=HOW=ˆ1,*-*;;]*.

Then we define an integer macro to save parameter 2 in the SIP call, with default value 9 *[ˆIM=MARG=ˆ2,9;;]*.

The left margin is increased by the value stored in MARG *[ˆLM=+ˆ$MARG;;]*

An integer macro is defined to save parameter 3 in the SIP call, in which default value is set to default paragraph spacing *[ˆIM=IPS=ˆ3,ˆ$PS;;;]*.

A macro is defined to save parameter 4 in the SIP call, with default value *R [ˆMD=IJM=ˆ4,R;;]*.

A macro is defined to save parameter 5 in the SIP call, with default empty parameter value *-* as default *[ˆMD=JUST=ˆ5,*-*;;]*.

Finally the current text line in the document is terminated *[ˆBL;]*. If there is none, the directive has no effect.

DETAILED DESCRIPTION OF SECTION 2.1 IN THE SIP MACRO

As mentioned previously, the entire section 2 is there for the purpose of defining the IP macro. This section defines the introductory part of the IP macro.

First the paragraph spacing is defined *[^BL=^$IPS;;]*.

Then the macro which is going to store the margin text for the paragraph is cleared *[^CM/MD-SAVP;]*, and parameter-1 in the IP call is stored in it. *[^MA/MD-SAVP/^PM=1;;]*.

DETAILED DESCRIPTION OF SECTION 2.2 IN THE SIP MACRO

This section consists of testing whether the left margin is wide enough to create an inverted paragraph. The minimum value is set to 2 *[^IF|^$LM;<2|]*. This test has a THEN part and an ELSE part. If the result of the test is TRUE, ie., if the margin is too small, the rest of section 2.2 is expanded. In the opposite case, sections 2.3, 2.4 and 2.5 are expanded.

The rest of section 2.2 consists of writing an error message on the screen *[^MS]*. This message contains the name of the IP macro, together with the parameter it was called with.

DETAILED DESCRIPTION OF SECTION 2.3 IN THE SIP MACRO

This section starts with a conditional form feed *[^CP=^$PF;;]*, ie., if there are fewer lines left on the page than the value of section footing ($PF), a new page will be started.

After that all the format parameters are stored *[^NF;]*.

Justification mode is set to the value defined for justification of the margin text *[^JM=^IJM;;]*.

Then the size of the left margin is stored in the integer macro IM-SAVP *[^IM-SAVP=^$LM;;]*.

DETAILED DESCRIPTION OF SECTION 2.4 IN THE SIP MACRO

This section first tests whether justification mode for the margin text is right (R) *[ˆIF|''ˆIJM;'='R'|]*. If this is so, *only* the next line in this section is expanded. In the opposite case the next line is *not* expanded, but the subsequent lines are.

In the case where margin text justification is right, the left margin is set to 0 *[ˆLM=0;]* and the right margin to the position the left margin had *[ˆRM=ˆ$TW;-ˆ$IM-SAVP;;]*. This means that the entire text line is limited to the part reserved for the margin text.

If margin text justification is not right, the section tests whether it is left (L) *[ˆIF|''ˆIJM;='L'|]*. If this is the case, *only* the next line in this section is expanded. In the opposite case the next line is *not* expanded, but the subsequent lines are.

If margin text justification is left, the left margin is reduced by the size defined for the margin text *[ˆLM=-ˆ$MARG;;]* and the right margin set to 0 *[ˆRM=0;]*. This means that the margin text is to be justified between the left margin and the right border on the sheet, ie., line length for the paragraph text plus the part reserved for the margin text.

If margin text justification is not left either, the section tests whether it is centered (C) or stretched (S)
[ˆIF|[''ˆIJM;'='C']OR[''ˆIJM;'='S']|]. If this is so, the rest of the section is expanded. If not, the section is terminated.

If the margin text justification is either centered or stretched, the left margin is reduced by the size reserved for the margin text *[ˆLM=-ˆ$MARG;;]* and the right margin set in the position of the left margin *[ˆRM=ˆ$TW;-ˆ$IM-SAVP;;]*. This means that the total text line is reduced to becoming only that part which has been reserved for margin text.

DETAILED DESCRIPTION OF SECTION 2.5 IN THE SIP MACRO

In sections 2.3 and 2.4 we have defined the various format parameters for justification of the margin text. This section therefore starts with justification of the margin text *[ˆMD-SAVP;]*.

When this has been done we test whether the margin text cuts across the left margin for the paragraph text. This can only happen if the right margin is 0, and this condition has therefore been included in the test *[ˆIF|(ˆ$RM;=0)AND(ˆ$CPOS;>ˆ$IM-SAVP;)]*.

If the test gives TRUE as result, ie., if the margin text does cut across the left margin for the paragraph, the line is broken to make the paragraph text start on a new line *[ˆBL;]*.

If not, the line is broken so that the paragraph text may start on the same line *[ˆBL=-1;]*, although further to the right, of course.

The section and the whole IP macro are then terminated by fetching the format parameters stored at the beginning of section 2.3.

DETAILED DESCRIPTION OF SECTION 3 IN THE SIP MACRO

We start, in this section, by testing whether it has been specified how the paragraphs initiated with the NP macro should be marked *[ˆIF|''ˆHOW;'='*-*'|]*. If this test gives a TRUE value, ie., if marking is not specified, the NP macro will be defined to call the IP macro with the text '- ' *[ˆMD/NP/ˆ<ˆIP=- ;ˆ>;]*.

If not, the NP macro will be defined to call the IP macro the with paragraph number ($IPNO) as a parameter, in the format specified and followed by ') ' *[ˆMD/NP/ˆ<ˆIP=ˆ$IPNO,ˆHOW;;) ;]*. In this case the NP macro will increment the paragraph number by one *[ˆIPNO=+1;]*.

DETAILED DESCRIPTION OF SECTION 4 IN THE SIP MACRO

This section tests whether justification mode for the paragraph text has been specified *[ˆIF|''ˆJUST;'<>'*-*'|]*. If it has, justification mode will be set to the value specified *[ˆJM=ˆJUST;;]*.

A sequence of inverted paragraphs started with the SIP macro must always be terminated with the EIP macro.

Definition:

```
^MD/EIP/^<^MK=IPNO;^MK=NP;^MK=HOW;^MK=JUST;^MK=IJM;
^LM=-^$MARG;;^BL=^$IPS;;^MK=MARG;^MK=IPS;^MK=IP;^OF;^>;
```

The expansion of this macro causes the current version of the following help macros defined in *SIP* to be killed: *IPNO*, *NP*, *HOW*, *JUST*, *IJM*, *MARG*, *IPS* and *IP*. The margin is set to the previous value, paragraph spacing is created, and previously stored format parameters are fetched again.

This means that when nested SIP levels are used, all macro definitions on the current SIP level are killed in order for the definitions on the previous SIP level to become valid again.

10.10 THE MACRO HEAD

The macro has been created for the purpose of making a simple version of the company's logo, to be used in letters, memos, etc. The macro uses a few global macros defined in the macro library. These have to be modified by the individual user organization, however.

FIRM	:	Resulting in the firm's name.
FADDRESS	:	Resulting in the firm's address.
FPHONE	:	Resulting in the firm's phone number.
FPLACE	:	Resulting in the firm's address.
FTELEX	:	Resulting in the firm's TELEX number.

Definition:

```
^MD/HEAD/^<^NF;^BT=+;^JM=L;^FM=NO;^HD,,-;^HD,,-;
^HU,,-;^aw,ooo   ooo ooooooo   ;
^HU,,-;^aw,oooo  ooo oooooooo  ; ^1/^<^FIRM;^>;
^HU,,-;^aw,ooooo ooo ooooooooo ;
^HU,,-;^aw,ooooooooo ooo   ooo ;
^HU,,-;^aw,ooooooooo ooo   ooo ; ^2/^<^FADDRESS;^BL=-1;^JM=R;^FPHONE;^>; ^JM=L;
^HU,,-;^aw,ooo ooooo ooooooooo ;
^HU,,-;^aw,ooo  oooo oooooooo  ;
^HU,,-;^aw,ooo   ooo ooooooo   ; ^3/^<^FPLACE;^BL=-1;^JM=R;^FTELEX;^>;
^JM=L;^HD,,-;^HD,,-;^HD,,-;^HD,,-;^HD,,-;^HD,,-;^OF;^>;
```

As you can see, this macro is rather simple.

You should make a note of a few things:

> When you use a combination of *HD* and *HU* directives without automatic return (parameter value '- ') you MUST use the same number of HU and HD directives on the same page of text. If you do not, the trailer will not be on the right line.

The use of *HD* and *HU* in this macro causes the line spacing in the logo itself to represent only one half of the value of current line spacing for the text itself.

The macro starts with the storage of format parameters *[^NF;]*. Bold text mode is activated *[^BT=+;]*. Justification and filling modes are defined *[^JM=L;^FM=NO;]*.

This is followed by two *HD* without return *[^HD,,-;^HD,,-;]*, which means that two half line feeds are carried out, but without the paper going back to the starting point. The number of *HD* to be included before and after the logo is in fact immaterial for the result of the logo itself, but it is of importance when it comes to placing the logo in relation to the document text.

We have now reached the first line of text in the logo, starting with half a line shift up, without return *[^HU,,-;]*. The text part is written with the *AW-directive [^aw ooo ooo ooooooo];* This causes all the characters to be considered as one word, so that no extra blanks will be inserted in the text during justification.

The next line of the logo is created in the same way, but is followed by additional text which is parameter 1 in the macro with the firm's name as default value: *[^1/^<^FIRM;^>;]*.

Lines 3 and 4 are created in the same way as line 1.

Line 5 in the logo is also followed by extra text, parameter 2 in the macro, with the firm's street address and phone numbers as default. These are left and right justified on the line, respectively, to the right of the logo text: *[^2/^<^FADDRESS;^BL=-1;^JM=R;^FPHONE;^>;]*

Justification mode is then set to left justification again *[^JM=L;]*.

Lines 6 and 7 are created in the same way as line 1.

Line 8 in the logo is also followed by extra text, parameter 3 in the macro. The firm's city/location and TELEX addresses are default. These are left and right justified on the line, respectively, to the right of the logo text:
[^3/^<^FPLACE^BL=-1;^JM=R;^FTELEX;^>;]

Justification mode is then set to left justification again.
[^JM=L;].

To counterbalance the 8 *HU directives* used to create the logo text, we now input 6 *HD-calls* (making a total of 8 when added to the 2 in the introduction).

The macro is terminated by resetting all format parameters to their original values *[^OF;]*.

If you want to modify this macro, and most of you probably will, you have to remember that there is only room for a limited number of alphanumerics of this size on a 'line', especially if you also want to include your firm's name and address, etc., as in our example.

10.11 THE MACRO MEMO

The macro is in the document macro category, and is used to write documents of the *MEMORAMDUM* type.

The macro may be called with up to 6 parameters:

1. To , to whom the memo is sent
2. Copy to (option) , who should receive copy of the memo
3. From , who sent the memo
4. Subject , the subject of the memo
5. Your ref.(option),
6. Our ref.(option)

Definition:

```
^MD/MEMO/^<^DX;^PH=0;^TB=0;
  ^HEAD/ / /^<_^FIRM;      M E M O_^>;^IR-SCH,MEMO;
  ^MD=SUBJECT=^4=^<^OP=Subject: ;^>;;
  ^JM=L;^FM=F;^SIP,,10,0;
  ^IP=To:       ;^IR-TON,1;^1=^<^OP=To:       ;^>;^IR-TOFF;^BL;
  ^IF|''2/*-*;'<>'*-*'|^<^IP=Copy to: ;^IR-TON,2;^>^2;^<^IR-TOFF;^>;^BL;
  ^RM=31;
  ^IP=From:       ;^IR-TON,3;^3=^<^OP=From:       ;^>;^IR-TOFF;^BL=-1;
  ^RM=0;^LM=+^AR.^$PW;*2/5;;
  ^IF|''5/*-*;'<>'*-*'|^<^IP=Your ref: ;^IR-TON,4;^>^5;^<^IR-TOFF;^>;^BL;
  ^IF|''6/*-*;'<>'*-*'|^<^IP=Our ref:    ;^IR-TON,5;^>^6;^<^IR-TOFF;^>;^BL=1;
  ^LM=-^AR.^$PW;*2/5;;
  ^IP=Date:      ;^IR-TON,6;^FDATE;^IR-TOFF;^BL=1;
  ^IP=Subject:       ;^IR-TON,7;_^SUBJECT;_^IR-TOFF;
  ^EIP;^H1=_^SUBJECT;_;^FM=C;^SI=0;^PI=0;^JM=S;^TL=contd....;
  ^PH=1;^TB=4;^BL=2;^EOD-STR=^<^TL= ;^>;^IR-TON,8;^>;
```

As you can see, the macro contains some directive call of the *^IR-* type. These are directives which prepare a conversion of the document for storage with *NOTIS-IR*. We shall not explaion these directives any further, since they are used independently of schema definition in NOTIS-IR. They will also be omitted from the explanation so as not to make it unclear.

This macro is relatively uncomplicated and straightforward, and we shall therefore give you a step-by-step explanation without splitting it into sections.

It starts with the specification that the document is for duplex copying: *[^DX;]*. The page header function is then turned off, and the top border set to 0: *[^PH=0;^TB=0;]*. This is done because the first page is to have a memo head instead of the usual page header.

A call to the logo macro is now executed. Parameters 1 and 2 are blank while parameter 3 becomes the firm's name with the additional text, *M E M O* underlined:
[^HEAD/ / /^<_^FIRM; M E M O_^>;].

The help macro SUBJECT is defined to contain parameter 4. If parameter 4 is unspecified, you will be asked for the value on the screen during formatting: *[^MD=SUBJECT=^4=^<^OP=Subject: ;^>;;]*.

Justification and filling modes are defined, and inverted paragraphs are prepared: *[^JM=L;^FM=F;^SIP,,10,0;]*.

The leading text 'To: ', with accompanying text from parameter 1, is written as an inverted paragraph. If parameter 1 is unspecified, you will be asked for the value on the screen during formatting:
[^IP=To: ;^1=^<^OP=To: ;^>;^BL;].

A test is now run to see whether parameter 2 is specified (this kind of test has been explained previously). If the parameter is specified, the leading text 'Copy to: ', and the parameter text, are written as inverted paragraphs. If parameter 2 is unspecified, the leading text is not written either:
[^IF|''^2/-*;'<>'*-*'|^<^IP=Copy to: ;^>^2;;*
].

The right margin is increased to make room for the reference texts: *[^RM=31;]*. Then the leading text 'From: ' is written, followed by parameter 3. If this parameter is unspecified, you will be asked for the value on the screen during formatting:
[^IP=From: ;^3=^<^OP=From: ;^>;].

The contents of this line in now written, WITHOUT linefeed: *[^BL=-1;]*. This means that subsequent text will be written on the same line.

Right margin is reset to 0, and the left margin is calculated as a function of the page width (it may of course be set as a constant):
*[^RM=0;^LM=+^AR.^$PW;*2/5;;]*.

'Your ref:' and 'Our ref:' are now written on separate lines relative to the new margins, *if they have been specified*. This is done in the same way as for parameter 2:
[ˆIF|''5/-*;'<>'*-*'|ˆ<ˆIP=Your ref: ;ˆ>ˆ5;;ˆBL;*
ˆIF|''6/-*;'<>'*-*'|ˆ<ˆIP=Our ref: ;ˆ>ˆ6;;ˆBL=1;].*

Left margin is now reset to its original value:
*[ˆLM=-ˆAR.ˆ$PW;*2/5;;].*

The leading text 'Date: ', followed by the date when the document was last updated is written as an inverted paragraph, followed by a blank line:
[ˆIP=Date: ;ˆFDATE;ˆBL=1;].

Then comes the leading text 'Subject: ' followed by the accompanying text, underlined, as an inverted paragraph:
[ˆIP=Subject: ;_ˆSubject;_].

The sequence of inverted paragraphs is terminated: *[ˆEIP;].*

Header-1 is defined to contain the subject text, underlined:
[ˆH1=_ˆSUBJCT;_;].

Filling and justification modes as well as section and paragraph indentations are defined: *[ˆFM=C;ˆSI=0;ˆPI=0;ˆJM=S;].*

Page trailer is defined to contain the text 'contd....':
[ˆTL=contd....;].

The page header function is activated again, and top border set back to the default value: *[ˆPH=1;ˆTB=4;].*

The MEO head is terminated with two blank lines: *[ˆBL=2;].*

Finally, the macro is terminated by defining *EOD-STR* (End-Of-Document STRing) to remove the page trailer. This means that all the pages except the last one will have 'contd....' as page trailer: *[ˆEOD-STR=ˆ<ˆTL= ;ˆ>;].*

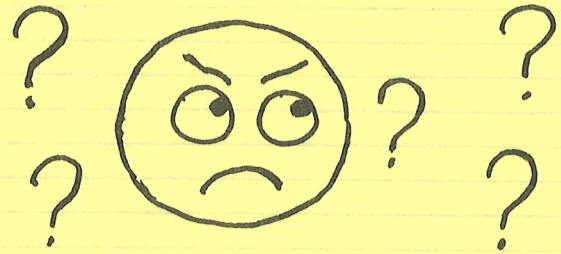
If you go through a document macro step by step in this way you will see that most of them are rather easily modified, especially if you only want to modify leading texts, etc.

GOOD LUCK!

LIST OF EXAMPLES

Index

* * * * * * * * * * SEND US YOUR COMMENTS!!! * * * * * * * * * *

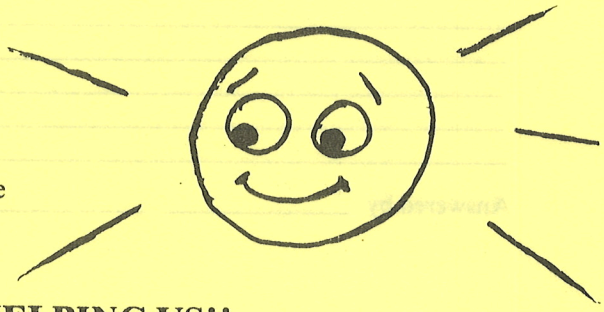


Are you frustrated because of unclear information in this manual? Do you have trouble finding things? Why don't you join the Reader's Club and send us a note? You will receive a membership card - and an answer to your comments.

Please let us know if you

* find errors
* cannot understand information
* cannot find information
* find needless information

Do you think we could improve the manual by rearranging the contents? You could also tell us if you like the manual!!



* * * * * * * * * HELP YOURSELF BY HELPING US!! * * * * * * * * *

Manual name: NOTIS-TF Macro Guide

Manual number: ND-63.009.01

What problems do you have? (use extra pages if needed) _____

Do you have suggestions for improving this manual? _____

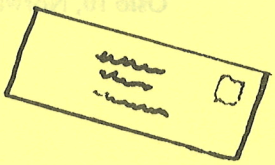
Your name: _____ Date: _____

Company: _____ Position: _____

Address: _____

What are you using this manual for? _____

Send to: Norsk Data A.S.
Documentation Department
P.O. Box 4, Lindeberg Gård
Oslo 10, Norway



Norsk Data's answer will be found on reverse side

Answer from Norsk Data _____

Answered by _____ _____ _____ _____ Date _____

Norsk Data A.S.
Documentation Department
P.O. Box 4, Lindeberg Gård
Oslo 10, Norway

Systems that put people first

NORSK DATA A.S JERIKOVN. 20 P.O. BOX 4 LINDEBERG GÅRD OSLO 10 NORWAY
TEL.: 02 - 30 90 30 - TELEX: 18661