GESStabs

User Manual

Gesellschaft für Software
in der Sozialforschung mbH

Waterloohain 6 - 8
22769 Hamburg
Tel.: +49 40 - 853 753 - 0
Fax: +49 40 - 853 753 - 33
www.gessgroup.de

Contents

I Abstract

***GESS* tabs** has been on the market since 1991. ***GESS* tabs** is tabulation software for the demanding professional. Due to its unique integration of typography, colour and basic graphs combined with elegant methods of evaluating single and multi punched variables, ***GESS* tabs** is a highly efficient analysis software for market and social research.

GESS tabs is a real "all-rounder": it can read and write several data formats, it is a flexible tool for data cleaning and data calculation, it masters iterative weightings, it can focus data from several sources into one (e.g. for panel data), it doesn't even forsake you if you need charts for a presentation! From simple net calculations up to the most complex analysis with significance tests, there is almost nothing that the user cannot achieve using ***GESS* tabs**.

Cross tables can present frequencies, percentage distributions, totals, percentage distribution of totals, variables, arithmetic, harmonic and geometric means etc. In addition profiles, simple frequency distributions and comparison of frequency distributions, column calculations, and error statistics are all possible.

Output:
The ***GESS*** tabulation system produces "ready for printing" tables of very high quality. ***GESS* tabs** is at the top of the list if output in high printing quality is required. Tables generated in Postscript become PDF-Data Files. At the same time tables in different formats can be additionally produced, e.g. in HTML format for direct web publishing, as CSV-Export or via the OLE interface directly to Microsoft Excel.

Additionally modified data files can be stored after calculating or cleaning: ***GESS* tabs** is a powerful and comfortable tool for data processing.

Program Commands:
All commands can be formulated using the simple-to-use instruction language, with a simple, format-free syntax. The ***GESS*** tabulation system provides special types of variables, including single and multiple answers, open questions, numeric questions or those with alpha-text elements. The command language is highly productive due to its macro capabilities.

Data Input:
GESS tabs can read ASCII files, SPSS-SAV files, and several kinds of column binary files, CSV files or dbase files. ASCII files can include numeric data and/or text. In order to work with SPSS-SAV files the relevant DLL(s) form SPSS (SPSSIO32.DLL etc.) is required.

Data Processing:
The range of data processing instructions includes instructions like `RECODE` or `GROUPRECODE` for re-coding, `COMPUTE` for calculations, `IF ... THEN ... ELSE` for logical processing, `IF... PRINT` and `IF ... ASSERT ...` for finding and erasing errors etc. For many cases there are special extensions for these instructions which make managing multi punch variables even more comfortable. Tables can of course be filtered in many ways.

Weighting:
During analysis, ***GESS tabs*** can calculate weighting-factors iteratively, or work with already given factors.

GESS tabs allows you to tabulate quickly and with attractive results:

Speed:

The use of **pre-processors** (macros) simplifies recurrent tasks.

Overcodes (and over-overcodes etc.) are automatically treated as part of a variable.

Codes from **open questions** are directly processed from text files, no additional processing of data is necessary.

Open questions from text files can be directly displayed **verbatim**.

According to user-defined marginal distribution ***GESS* tabs** calculates **case-weights** automatically, on the fly or they can be stored in an output file.

Special variable types and instructions for **multi-punches** simplify tabulation.

***GESS* tabs** calculates **significance tests** and displays them in the tables.

Information from **additional data sources** (e.g. panel information) can be used directly using an index key. It is not necessary to produce a modified output-file.

There are smart instructions within the ***GESS* tabs** instruction language to answer most of the common problems arising in market research.

Appearance:

***GESS* tabs** simply produces attractive and highly-legible tables. This is due to the **sophisticated typographic commands**

Important information can be directly highlighted, e.g. using **colour**.

There are **no text restrictions** for labels or questions, the text length is unlimited.

***GESS* tabs** can design tables according to **corporate identity**, e.g. including logos.

Tabulation:

Complex tables are easy to produce from many variables.

Dummy cross tables are easy to produce**,** e.g. in order to display several products next to each other in one header (monadic, semi-monadic).

One table with many items or a long header can easily be displayed over **several pages**.

***GESS* tabs** can calculate and present a huge variety of statistics (n>50).

Several statistics can be shown within one data cell (e.g. absolute frequency, percentage, mean)

Rows or columns within a table can be based on different weights and tables can present **weighted and unweighted results** side by side or underneath each other.

Results within tables can be **sorted**, e.g. descending order according to frequency or even hierarchy according to overdcodes.

Tables can be transferred directly to Excel via OLE; in this case graphics can also be produced automatically in EXCEL.

***GESS* tabs** is able to produce tables directly in **HTML** for easy web presentation in all browsers.

A First Example

Using a data file called DADA.DAT for this simple example we will assume that age has been coded in five groups in column 16 and that five fictitious regions (central, north to west) have been recoded in column 43. For our first example ***GESS* tabs** will produce a table showing the regions in the header and the age groups on the left-hand side. The table is to show just the frequency percentages in columns.

```
{1}    DATAFILE = dada.dat;
{2}    PRINTFILE PS = mini.ps ;
{3}    INCLUDE = EINFACH.FMT;
{4}    SINGLEQ Alter = 16
       LABELS
       1 "18#24"
       2 "25#30"
       3 "31#45"
       4 "46#60"
       5 "61 and älter";
       SINGLEQ Bezirk = 43
       LABELS
       1 Mitte
       2 Nord
       3 Süd
       4 Ost
       5 West;
{5}    TABLETYPE = COLUMNPERCENT;
{6}    TABLE = Bezirk BY alter;
{7}    END;
```

Tabelle 1:

Sp. %	Insgesamt	Bezirk				
		Mitte	Nord	Süd	Ost	West
Zahl der Befragten (abs.)	199	24	46	60	48	21
Alter						
18-24	12	13	15	5	15	14
25-30	29	21	26	32	33	24
31-45	24	29	20	25	21	29
46-60	24	33	24	27	21	10
61 und älter	13	4	15	12	10	24

GESS mbH *

Here is the result. ***GESS* tabs** has produced a standard header, "Table 1:“, as well as the standard texts "Total" and "Number of Cases" which can be easily altered by the user. In the first line of the table ***GESS* tabs** has inserted a row with the absolute values which is standard for "COLUMNPERCENT" tables. This line reads as follows: In the Central District 24 people were questioned, etc. The first column of the table shows the total presenting all the age groups vertically which is also standard for the "COLUMNPERCENT" table. The remaining columns contain the percentages of the different age groups in the five districts.

Below is a short explanation of the individual steps in the program (the steps are the numbers in curly brackets in the example):

1. The file is called `DADA.DAT`.
2. A file called "`mini.ps`" is to be used for printing using a Postscript printer.
3. A format-file is included which defines a "simple" table layout.
4. The variable "age" is defined which is coded in column 16 of the file and the classification of the variables are to be provided with texts, so-called `LABELS`. Then the variable "District" is dealt with in the same way.
5. All succeeding tables are designated as column percentages.
6. The table is designated as "District" against "Age".
7. The end of the program is reached.

Example with Automatic Weighting

This second example demonstrates iterative weighting and special types of variables in order to show how ***GESS* tabs** solves the problem of multiple responses. Firstly, the weighting: our data file contains the variables "Age" (codes 1-3), "Sex" (codes 1-2) and a regional variable "District" (codes 1-5). The two-dimensional distribution of age and sex and a distribution by region as weights have already been provided for the purpose of weighting.

	Geschlecht		Bezirk	
	männ-lich	weib-		
Alter			Mitte	12.0
			Nord	23.3
			Süd	30.1
bis 24 Jahre	14.3	19.0	Ost	24.1
24 bis 50 Jahre	22.1	24.1	West	10.5
über 50 Jahre	12.5	8.0		

Producing the `WEIGHTCELLS` for the variation between the districts is the simplest task as it is only necessary to define which percentage values are valid for the five districts. The relevant part of our instruction file should look something like this:

```
WEIGHTCELLS Bezirk =
1 : 12.0%
2 : 23.3%
3 : 30.1%
4 : 24.1%
5 : 10.5%;
```

A variable for the combination between age and sex needs to be defined first and then the program is continued as above**.**

```
COMPUTE AgeSex = Alter * 10 + Geschlecht;
WEIGHTCELLS AgeSex =
11 : 14.3%  12 : 19.0%
21 : 22.1%  22 : 24.1%
31 : 12.5%  32 :  8.0%;
```

These instructions are sufficient in order to carry out iterative weighting during an analysis run using the given distributions. For clarity the instructions are organised in the same order as the age groups. Nothing else is necessary!

In our example questionnaire there is also a section of questions asking what means-of-transport are used to travel on holiday. Naturally it is possible to use more than one form-of-transport. For the survey the question can be answered by using a series of YES/NO alternatives, often in the short form of ticking the relevant boxes. This type of multiple response variable is presented in ***GESS* tabs** as a group of dichotomic variables with the key word `DICHOQ` (or formerly `GROUPVAR`). The columns must be named and the numeric codes showing group membership must be designated. The instruction for this is:

```
DICHOQ Verkehrsmittel =
20  "Auto"
 *  "Flug-zeug"
 *  "Eisen-bahn"
 *  "Fahr-rad"
 *  "Mo-tor-rad"
 *   "ande-res Ver-kehrs-mittel"
EQ 1;
```

Furthermore, questions were asked about holiday destinations using first, second, and third choice. The totals for each destination are to be added together, regardless in which order they were chosen. For this purpose ***GESS* tabs** provides the user with a multiple response variable type `MULTIQ`.

```
MULTIQ Urlaubsziele = 40 3
LABELS
1 "in diesem unserem Land"
2 Österreich
3 Italien
4 Spanien
5 "andere Gegend";
```

The entries one to three are coded in columns 40, 41 and 42 (starting at column 40, field length 3).

The banner at the top of the table should show the means-of-transport used, the column furthest left the holiday destinations. The row and column percentages are to be written in the table cells and the average amount of money spent by the people questioned is also to be shown at the edge of the table. This variable is taken as an amount directly from the data record.

Firstly, the cell elements are defined and in this example the cell contents and frame elements are defined explicitly. (The pre-defined `TABELETYPEs` which were used in the first example are a simpler alternative for table types that are used repeatedly).

```
CELLELEMENTS = ROWPERCENT, COLUMNPERCENT;
FRAMEELEMENTS = TOTALROW, TOTALCOLUMN, ABSROW, ABSCOLUMN;
```

The `CELLELEMENTS` statement defines that both the column and row percentages are shown. The `FRAMEELEMENTS` statement defines a complicated table-frame which should contain, both horizontally and vertically, the number of respondents and the relevant "totals".

The type of format and the use of a standard text have to be defined in order to organise the output. The row and column percentages are provided with a percentage sign, and instead of the long-winded "Number of Respondents", there is just an "n".

```
FORMAT COLUMNPERCENT = "# %";
FORMAT ROWPERCENT = "# %";
CASESTITLE = "n";
```

Now there is only the table itself left to define:

```
TABLE = Verkehrsmittel MEAN( Money ) BY Urlaubsziele MEAN( Money );
```

Firstly the contents of the head of the table ("stub") have to be defined in the `TABLE` statement: the forms-of-transport should be shown followed by a column containing the averages for "holiday budget". The contents of the left-hand column ("banner") are defined after the key word `BY`. Here everything is symmetrically the same: first the holiday destinations, then the rows with the averages of the "holiday budget".

The complete program for the table is printed out below:

```
DATAFILE = bsp2.dat;
PRINTFILE POSTSCRIPT = bsp2.lst;
INCLUDE FMT01.FMT;
DOCUMENT = "Example 2";

SINGLEQ Bezirk = 16;
WEIGHTCELLS Bezirk =
1 : 12.0%
```

```
2 : 23.3%
3 : 30.1%
4 : 24.1%
5 : 10.5%;

SINGLEQ Alter = 17;
SINGLEQ Geschlecht = 1;
COMPUTE AgeSex = Alter * 10 + Geschlecht;
WEIGHTCELLS AgeSex =
11 : 14.3%        12 : 19.0%
21 : 22.1%        22 : 24.1%
31 : 12.5%        32 : 8.0%;

DICHOQ Verkehrsmittel =
20  "Auto"
 *  "Flug-zeug"
 *  "Eisen-bahn"
 *  "Fahr-rad"
 *  "Mo-tor-rad"
 *   "ande-res Ver-kehrs-mittel"
EQ 1;

MULTIQ Urlaubsziele = 40 3
LABELS
1 "in diesem unserem Land"
2 Österreich
3 Italien
4 Spanien
5 "andere Gegend";

SINGLEQ Money = TITLE "Urlaubs-kasse (DM)" 43 4;

FORMAT COLUMNPERCENT = "# %";
FORMAT ROWPERCENT = "# %";
CASESTITLE = "n";

TABLETITLE = "Tabelle 2: Verkehrsmittel and Reiseziele";
CELLELEMENTS = ROWPERCENT, COLUMNPERCENT;
FRAMEELEMENTS = TOTALROW, TOTALCOLUMN, ABSROW, ABSCOLUMN;
TABLE = Verkehrsmittel MEAN( Money ) BY Urlaubsziele MEAN( Money );

END;
```

Tabelle 2: Verkehrsmittel und Reiseziele

Zeile % Sp. %	n	Insgesamt	Verkehrsmittel: Auto	Flugzeug	Eisenbahn	Fahrrad	Motorrad	anderes Verkehrsmittel	Urlaubskasse (DM)
n	512		318	161	134	58	60	67	512
Insgesamt		100 % 100 %	62 % 100 %	32 % 100 %	26 % 100 %	11 % 100 %	12 % 100 %	13 % 100 %	**3365**
Urlaubsziele									
in diesem unsrem Land	183	100 % 36 %	58 % 33 %	32 % 36 %	28 % 38 %	11 % 35 %	10 % 30 %	16 % 44 %	**3377**
Österreich	299	100 % 58 %	66 % 62 %	29 % 54 %	26 % 57 %	12 % 62 %	11 % 57 %	14 % 64 %	**3422**
Italien	281	100 % 55 %	56 % 50 %	33 % 58 %	29 % 60 %	12 % 58 %	13 % 61 %	14 % 57 %	**3291**
Spanien	288	100 % 56 %	65 % 59 %	32 % 56 %	23 % 49 %	13 % 64 %	12 % 58 %	12 % 53 %	**3335**
andere Gegend	185	100 % 36 %	63 % 37 %	33 % 37 %	23 % 31 %	10 % 32 %	8 % 26 %	12 % 34 %	**3389**
Urlaubskasse (DM)	512	**3365**	**3404**	**3379**	**3422**	**3606**	**3402**	**3184**	

GESS mbH — Beispiele

A standard format file is included: `FMT01.FMT` for the graphical layout. Due to the narrow, proportional writing, this is a format that can produce tables with many cells. A few of the particular abilities of this format file should be explained. For the column and row percentages different fonts are used (Times-Roman and Helvetica-Narrow), as for the means (**Helvetica-Narrow-Bold**). Additionally the data in the crosstab is lightly shaded grey (see `SHADE` instructions), whereas the data at the edge of the table is not in order to increase clarity.

NB: The figures in the table are purely hypothetical figures made up for the sake of demonstration.

Example: Where are Interesting Differences?

When starting the first analysis of data it can be useful for the software to point out where codes differ significantly from the norm. Many steps of data analysis are based on revealing effects this way.

***GESS* tabs** provides the user with a simple-to-use tool for this: The command `MARKCELLS` causes all cells in a cross table to be examined as to whether they vary significantly from the total. The deviance from the total is highlighted in different colours defined by the user according to the direction of the deviance, e.g. deviance above total = blue, below = red.

Hamburg 2008: Wahlverhalten nach Geschlecht und Alter

Wem gaben Sie Ihre Stimme für die LANDESLISTE?
(gelber Stimmzettel)

		Geschlecht		Altersgruppen				
Sp. %	TOTAL	männlich	weiblich	18 - 24 Jahre	25 - 34 Jahre	35 - 44 Jahre	45 - 59 Jahre	60 und älter
Basis	38632	17962	19680	2547	5475	7216	8794	13670
CDU	42,2	41,3	43,4	31,5	37,4	37,1	36,6	52,0
SPD	33,8	32,2	35,1	40,7	35,0	33,1	34,4	32,1
Grüne	9,5	8,7	10,3	11,0	12,8	15,1	12,4	3,3
FDP	4,7	5,9	3,5	4,1	5,3	4,4	4,2	5,0
Linke	6,3	7,6	5,0	5,9	5,5	6,5	9,7	4,4
Sonstige	2,6	3,4	1,7	6,2	3,6	3,2	1,8	1,7
Keine Angabe	0,9	0,9	1,0	0,5	0,5	0,6	0,9	1,5

Rote Felder = signifikant unter dem Total, blaue Felder = signifikant über dem Total (0,1%-Niveau)

Software by GESS

Some of the aspects of electoral behaviour can now be recognised at once due to the different colours. Old hands at electoral sociology will not be surprised at these results but it is still impressive how quickly the table can be interpreted if such assistance is available, e.g. the CDU has a lot more sway with the older generations, but in all other age-groups they received below-average votes, Green and SPD reached below average in the over 60's and the SPD is only above average in the very young voters, the Greens with the 25-59 year olds. FPD, Left and other parties have in common that receive below average votes from women etc. etc.

The commands for the table are astonishing in their simplicity (`#k0` refers to the variables sex and age):

```
Table =  #k0   by  Lzweit;
bottomtext =
"Rote Felder = significant unter dem Total, blaue Felder = significant
über dem Total";
```

The colouring of the deviance upwards or downwards is easily defined using the `MARKCELLS` statement which expects six colour codes: firstly the three for the cells below the average, then the three for those above the average. The three colour codes are valid for the significant levels 5%, 1% and 0.1%. RGB-colour codes are defined in accordance with the Postscript colour models 0 - 1.

```
RGB = YES;
MARKCELLS = YES COLOR
1 1 1          // %5-Niveau          Kleiner als der Durchschnitt
1 1 1          // 1%-Niveau          Kleiner als der Durchschnitt
1 0.80 0.80    // 0,1%-Niveau        Kleiner als der Durchschnitt
1 1 1          // %5-Niveau          Größer als der Durchschnitt
```

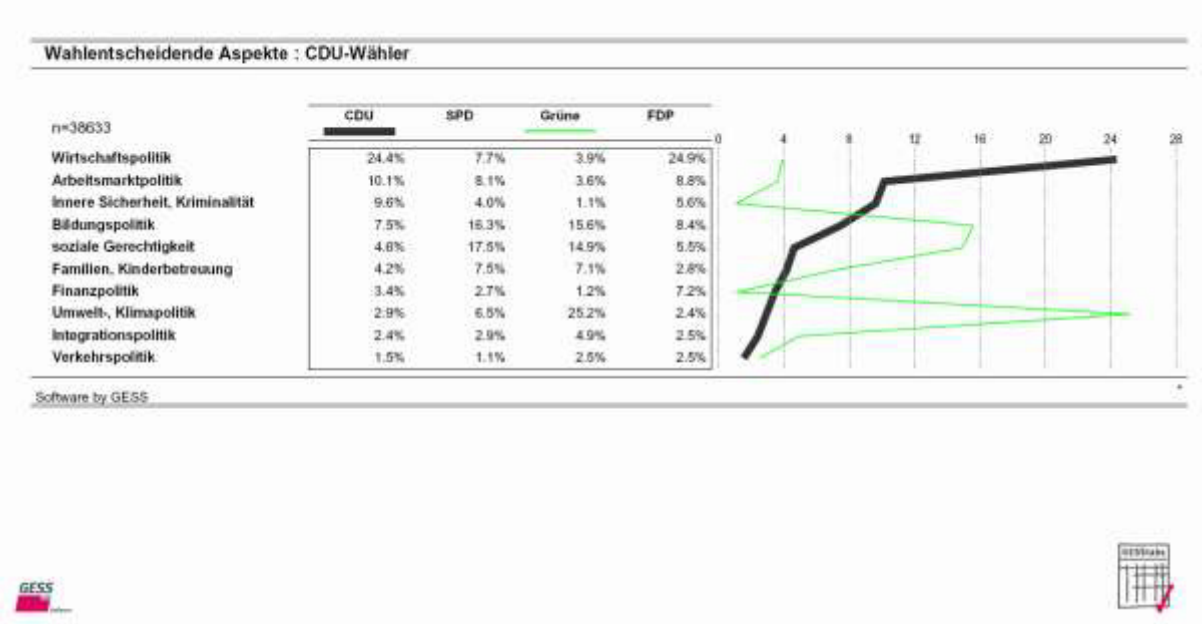
```
      1 1 1          // 1%-Niveau          Größer als der Durchschnitt
      0.80 0.80 1    // 0,1%-Niveau        Größer als der Durchschnitt
;
```

Example: Summary Tables with Graphics

We shall stay at the 2008 election in Hamburg for a while. It resulted in the first coalition between the CDU and the Greens in a county parliament. The Hamburg electorate were asked in a post-election survey what had been important criteria for their voting behaviour; this question allowed multiple responses. A glance at the answers to this question in the survey by a television station (ARD Germany) shows the very varied motivations of the voters for these two parties.

The following `PROFILE` table shows the percentages for the four main parties and ten pre-defined political aspects which were said by the voters to be important for their voting behaviour

This table has been sorted according to the number of CDU responses. The graphics do not include the voters for the SPD or FDP in order to highlight the comparison between the voters for the resulting coalition.



And how is this achieved? below is the relevant section of the program:

```
#expand #black 0.2 0.2 0.2
#expand #red 1 0 0
#expand #green 0 1 0
#expand #yellow 1 1 0

RGB = YES;

TABLETITLE = "Wahlentscheidende Aspekte : CDU-Wähler";
PROFILELINES =
| 1 : PATTERN 5 COLOR #black WIDTH 5
| 2 : HIDDEN
| 3 : PATTERN 5 COLOR #green
| 4 : HIDDEN
;
PROFILESORT = 1 DESCEND;
PROFILE =
aspekt1
aspekt2
aspekt3
aspekt4
aspekt5
```

```
aspekt6
aspekt7
aspekt8
aspekt9
aspekt10
BY Partei;
```

First the colour codes for the four interesting parties are simply defined using the `#expand` statement, e.g. as a result the command `#black` at later positions is replaced by the three numbers `0.2 0.2 0.2` which in RGB produces a very dark grey. The same is done for the other parties.

The properties of the line are defined before the `PROFILE` statement, the lines for the SPD and the FDP remain hidden in the name of clarity and are thus defined as `HIDDEN`. The codes of the `PROFILELINES` refer to the codes of the `BY` variable, in this case the party. The CDU receives a particularly thick line using the `WIDTH` statement. These profile comparisons are easier to read if they are sorted by one dimension, in this case the votes for the CDU (`Partei=1`).

Example: Calculation and Table ADD

But in market research only a few tables are as easy to generate as these first examples. The surveys and thus the data files are usually more complex and the tables are required to summarise values from several variables.

Let us use the following example: 250 respondents were questioned about their last 3 mobile phones before additional questions were asked about two of these mobiles, including the memory capacity. This is to be shown against a stub of mobile phone makes.

There are three variables, `Handy1`, `Handy2` and `Handy3` which contain the makes. There are the variables `Steuer1` and `Steuer2` for the additional question blocks regarding two of the mobiles which contain the three mobiles described, the first, second or third named.

Two variables thus need to be calculated from the values of `Handy1` to `Handy3` and `Steuer1` and `Steuer2`: the make of the first and the second mobiles named.

This could be achieved using different `IF` statements, it is however easier to use the special command `INDEXVAR`. The `INDEXVAR` command contains a numeric control variable and a variable list from which a value is chosen using the control variable.

In our example: if `Steuer1` contains the value 3 (="`3. Handy`") then in the first question block `(Marke_1)` the content of the variable `Handy3` is the make of the mobile responded to.
The `INDEXVAR` provides this compact solution:

```
INDEXVAR Marke_1 = Handy1 Handy2 Handy3 by Steuer1;
INDEXVAR Marke_2 = Handy1 Handy2 Handy3 by Steuer2;
```

As these variables have the same characteristics or labels as `Handy1`, `Handy2` and `Handy3` these variables can also have the same labels. This is most cleverly achieved using a `COPY` command as all later changes are automatically carried over. As far as possible it is sensible to avoid redundancy.

```
COPYLABELS Marke_1 Marke_2 = Handy1
```

The memory capacity of the mobile is available in the form of two variables with the size variations designated by the client: `g_sto_1` and `g_sto_2`. The required table is thus actually made up of two tables; firstly `Marke_1` against `g_sto_1` and then `Marke_2` against `g_sto_2`. These tables are then added together.

And this can be achieved by ***GESS* tabs** like so:

```
TABLE          = Marke_1 by g_sto_1;
TABLE ADD      = Marke_2 by g_sto_2;
```

Tabelle 6
Q02: Wieviel Speicher steht zur Verfügung

Abs. Sp. %	Insgesamt	Marke				
		Nokia	Samsung	Siemens	Blackberry	Apple
n (ungewichtet)	500	72	170	98	117	43
n (gewichtet)	500	68	171	97	119	44
1 bis 256 MB	64 12,8 %	6 8,3 %	22 13,0 %	14 14,1 %	12 10,2 %	10 23,0 %
257 bis 512 MB	77 15,4 %	15 21,9 %	36 21,3 %	10 10,2 %	6 4,7 %	10 23,0 %
513 bis 1024 MB	122 24,4 %	13 18,5 %	32 18,4 %	20 21,1 %	54 45,7 %	3 6,6 %
1 bis 2 GB	69 13,9 %	6 8,3 %	33 19,2 %	13 13,9 %	12 9,8 %	6 12,8 %
2 bis 4 GB	168 33,6 %	29 43,0 %	48 28,2 %	40 40,8 %	35 29,6 %	15 34,7 %
Gesamt (alle Fälle)	500 100,0 %	68 100,0 %	171 100,0 %	97 100,0 %	119 100,0 %	44 100,0 %

Signifikant vom Durchschnitt abweichende Zellen sind eingefärbt: blau über dem Durchschnitt, rot darunter

Software by GESS

Additionally the memory capacity is provided as a numerical position in megabytes (`sto_1` and `sto_2`), and we can add a mean:

```
TABLE     = Marke_1 by g_sto_1 MEAN :DESCRIPTION 'Mittelwert' ( sto_1
);
TABLE ADD = Marke_2 by g_sto_2 MEAN :DESCRIPTION 'Mittelwert' ( sto_2
);
```

In order to demonstrate how easily logos can be integrated into the tables, the ***GESS*** logo has been added four times. The instructions are as follows:

```
EPS = "gessnewlogo.eps" 45 55 WIDTH 80;
EPS = "gessnewlogo.eps" 120 55 WIDTH 55;
EPS = "gessnewlogo.eps" 180 55 WIDTH 40;
EPS = "gessnewlogo.eps" 230 55 WIDTH 28;
```

Tabelle 6
Q02: Wieviel Speicher steht zur Verfügung

Abs. Sp. %	Insgesamt	Marke				
		Nokia	Samsung	Siemens	Blackberry	Apple
n (ungewichtet)	500	72	170	98	117	43
n (gewichtet)	500	68	171	97	119	44
1 bis 256 MB	64 12,8 %	6 8,3 %	22 13,0 %	14 14,1 %	12 10,2 %	10 23,0 %
257 bis 512 MB	77 15,4 %	15 21,9 %	36 21,3 %	10 10,2 %	6 4,7 %	10 23,0 %
513 bis 1024 MB	122 24,4 %	13 18,5 %	32 18,4 %	20 21,1 %	54 45,7 %	3 6,6 %
1 bis 2 GB	69 13,9 %	6 8,3 %	33 19,2 %	13 13,9 %	12 9,8 %	6 12,8 %
2 bis 4 GB	168 33,6 %	29 43,0 %	48 28,2 %	40 40,8 %	35 29,6 %	15 34,7 %
Gesamt (alle Fälle)	500 100,0 %	68 100,0 %	171 100,0 %	97 100,0 %	119 100,0 %	44 100,0 %
Mittelwert Verfügbarer Speicher in MB	1651,4	1839,3	1572,4	1731,7	1605,0	1617,9

Signifikant vom Durchschnitt abweichende Zellen sind eingefärbt: blau über dem Durchschnitt, rot darunter

Software by GESS

Again the MARKCELLS statement has come into play; in this case the data cells have already been coloured at a deviance of 5%. MARKCELLS reacts only to frequency; deviance in the means are not coloured.

II Instructions for Table Specifications

In practice most tables required are cross tables and most of the specifications for cross tables are designated using the key word TABLE which can generate many different types of tables.

A series of special tables are easier to generate using the syntax of the XTAB statement described further below. It produces the same tables but is easier to use.

Additionally there are further table commands; COMPARE, PROFILE, COLUMNCOUNT, CODEBOOK and SUMMARY.

There are a number of commands (QUALITAB) for tabulating qualitative data with few case numbers to EXCEL where data for individual cases is often required which convert data. There is separate documentation for this in WEB.

TABLE

Following are examples of how different tables can be generated using the TABLE statement. First of all the variables bones are documented in the file var.inc which we are using for our examples:

```
variable geschl = * 1 labels
1 "Männlich"
2 "Weiblich"
;
title="Geschlecht";
Text="
Geschlecht des Befragten
";
helptext="
not fragen: Interviewereinschätzung
";
nomissing=yes;

variable alter = * 1 labels
1 "18-29 Jahre"
2 "30-49 Jahre"
3 "50-64  Jahre"
;
title="Alter";
text="
Alter des Befragten
";
nomissing=yes;

familyvar bekannt = * 9 1 labels
1 "Alcatel"
2 "Nokia"
3 "Motorola"
4 "Panasonic"
5 "Sagem"
6 "Samsung"
7 "Siemens"
8 "Sony-Ericsson"
9 "keine davon" single always
;
```

```
title="Bekannte Marken von Mobiltelefonen";
text="
Welche dieser Mobiltelefone kennen Sie?
";
nomissing=yes;

familyvar gekauft = * 9 1 labels
1 "Alcatel"
2 "Nokia"
3 "Motorola"
4 "Panasonic"
5 "Sagem"
6 "Samsung"
7 "Siemens"
8 "Sony-Ericsson"
9 "keine davon" single always bottom
;

title="Gekaufte Marken von Mobiltelefonen";
text="
Welche dieser Mobiltelefone haben Sie sich schon einmal gekauft?
";
nomissing=yes;

variable item_1 = * 1 labels
1 "stimme ganz and gar nicht zu"
2 "stimme eher nicht zu"
3 "weder/noch"
4 "stimme eher zu"
5 "stimme ganz and gar zu"
;
title="Die Farbe ist mir wichtig";
text="
Bitte stufen Sie dieses Item ein:
Die Farbe ist mir wichtig

";

variable item_2 = * 1 labels
1 "stimme ganz and gar nicht zu"
2 "stimme eher nicht zu"
3 "weder/noch"
4 "stimme eher zu"
5 "stimme ganz and gar zu"
;
title="Eine gute Sprachqualität ist mir wichtig";
text="
Bitte stufen Sie dieses Item ein:
Eine gute Sprachqualität ist mir wichtig

";

variable item_3 = * 1 labels
1 "stimme ganz and gar nicht zu"
2 "stimme eher nicht zu"
3 "weder/noch"
4 "stimme eher zu"
5 "stimme ganz and gar zu"
```

```
;
title="Eine gute Ausstattung ist mir wichtig";
text="
Bitte stufen Sie dieses Item ein:
Eine gute Ausstattung ist mir wichtig

";

variable item_4 = * 1 labels
1 "stimme ganz and gar nicht zu"
2 "stimme eher nicht zu"
3 "weder/noch"
4 "stimme eher zu"
5 "stimme ganz and gar zu"
;
title="Ich bevorzuge generell eine bestimmte Marke";
text="
Bitte stufen Sie dieses Item ein:
Ich bevorzuge generell eine bestimmte Marke

";

variable item_5 = * 1 labels
1 "stimme ganz and gar nicht zu"
2 "stimme eher nicht zu"
3 "weder/noch"
4 "stimme eher zu"
5 "stimme ganz and gar zu"
;
title="Ich lehne eine bestimmte Marke generell ab";
text="
Bitte stufen Sie dieses Item ein:
Ich lehne eine bestimmte Marke generell ab

";
random = item_1 item_2 item_3 item_4 item_5;
```

The easiest example is to generate a table where two of the variables defined in `EXAMPLE.INC` are tabulated against each other: age in the stub and the question "known" in the banner. To do this we write the following statement with the editor in the file `TAB.INC`:

```
TABLE = alter BY bekannt;
```

This is the simplest form of ***GESS* tabs** table statement. The variable to be shown in the stub comes after the key word `TABLE` followed by the key word `BY` and then the variable for the banner. The command is finished off with a semicolon.

The `TABLE` command uses the key word "`BY`" to separate the table between the X-Axis (or the stub) and the Y-Axis (or the banner). The characteristics of the variable "age" will thus appear in the X-Axis in the stub; the characteristics of "known" thus in the Y-Axis in the banner. Thus the elements for the stub are defined first and then those of the banner.

Tabelle 1

Abs.	Alter		
	18-29 Jahre	30-49 Jahre	50-64 Jahre
N	162	157	163
Bekannte Marken von Mobiltelefonen			
Alcatel	84	93	94
Nokia	96	101	95
Motorola	94	103	94
Panasonic	99	90	96
Sagem	98	101	108
Samsung	97	103	106
Siemens	88	102	87
Sony-Ericsson	95	89	108
keine davon	24	13	10

GESS mbH *

GESS

The TABLE statement covers somewhat over 80 different so-called CELLELEMENTS or contents which can be presented in individual cells. We have made no other choice so ***GESS* tabs** has presented the results as absolute frequency (in the case of weighted data this would be the total of the relevant case weights).

Additionally different elements of the table frame can be defined, e.g. an additional Total column or a row with relevant absolute basis. This is done using the ***GESS* tabs** command FRAMEELEMENTS.

GESS TABLE is made up of three optional frame rows or columns which are defined using the FRAMEELEMENTS command. The three types of rows are as follows:

```
TOTALROW          ABSROW          PHYSICALROW
```

and these are the three types of columns:

```
TOTALCOLUMN       ABSCOLUMN       PHYSICALCOLUMN.
```

TOTALROW and TOTALCOLUMN have the same standard as has been defined for the cell elements in the table body, although they can be further modified (see the description of the use of FRAMEELEMENTS).

ABSROW and ABSCOLUMN show the absolute frequency. NB: if using weighted data the total is of the rounded sum of the weights.

PHYSICALROW and PHYSICALCOLUMN show the unweighted frequency; if using unweighted data both totals are identical to their absolute equivalents.

In the standard case of a table with absolute frequency there is only one of the six frame elements in use; the row with absolute frequency: ABSROW.

In our case column percentages are required, i.e. CELLELEMENTS COLUMNPERCENT. A total column and an absolute row would go nicely with this.

```
CELLELEMENTS = COLUMNPERCENT;
```

```
FRAMEELEMENTS = ABSROW TOTALCOLUMN;
TABLE = alter BY bekannt;
```

Tabelle 2

Col %	Total	Alter		
		18-29 Jahre	30-49 Jahre	50-64 Jahre
N	482	162	157	163
Bekannte Marken von Mobiltelefonen				
Alcatel	56.2 %	51.9 %	59.2 %	57.7 %
Nokia	60.6 %	59.3 %	64.3 %	58.3 %
Motorola	60.4 %	58.0 %	65.6 %	57.7 %
Panasonic	59.1 %	61.1 %	57.3 %	58.9 %
Sagem	63.7 %	60.5 %	64.3 %	66.3 %
Samsung	63.5 %	59.9 %	65.6 %	65.0 %
Siemens	57.5 %	54.3 %	65.0 %	53.4 %
Sony-Ericsson	60.6 %	58.6 %	56.7 %	66.3 %
keine davon	9.8 %	14.8 %	8.3 %	6.1 %

GESS mbH *

GESS

The table looks the way it does thanks to the format file. The format file contains:
e.g.

```
format columnpercent = "##.# %";
```

This format defines the presentation of the column percentages with a percent sign after a space and one decimal place. The relevant fonts are also defined in the format file,
e.g.

```
usefont labels X    = "Helvetica-Bold" size 9;
```

Usually however the stub does not contain only one simple variable such as three age-groups so the next example shows a stub with two variables. In most cases the stub is defined using `#expand` as it is the easiest way to change a stub for a whole set of tables by merely changing the content of `#expand`. If an `#expand` has been defined and is found at a later point in the script the text of the `#expand` command is replaced by the series of signs.

```
#expand #k geschl alter
TABLE = #k BY bekannt;
```

thus has the same effect as

```
TABLE = geschl alter BY bekannt;
```

And this is the result:

Tabelle 3

Col %	Total	Geschlecht		Alter		
		Männlich	Weiblich	18-29 Jahre	30-49 Jahre	50-64 Jahre
N	482	255	227	162	157	163
Bekannte Marken von Mobiltelefonen						
Alcatel	56.2 %	54.1 %	58.6 %	51.9 %	59.2 %	57.7 %
Nokia	60.6 %	58.0 %	63.4 %	59.3 %	64.3 %	58.3 %
Motorola	60.4 %	61.2 %	59.5 %	58.0 %	65.6 %	57.7 %
Panasonic	59.1 %	55.3 %	63.4 %	61.1 %	57.3 %	58.9 %
Sagem	63.7 %	64.3 %	63.0 %	60.5 %	64.3 %	66.3 %
Samsung	63.5 %	59.2 %	68.3 %	59.9 %	65.6 %	65.0 %
Siemens	57.5 %	54.1 %	61.2 %	54.3 %	65.0 %	53.4 %
Sony-Ericsson	60.6 %	61.6 %	59.5 %	58.6 %	56.7 %	66.3 %
keine davon	9.8 %	11.8 %	7.5 %	14.8 %	8.3 %	6.1 %

GESS mbH *

GESS

Instead of having the variable title of the variable "known" above the labels we would rather have the question text. In order to achieve this we must 1. disable the `VARTITLE` and 2. tell ***GESS* tabs** where to find the question text.[1]

While we are improving the presentation we should also provide the tables with a more informative title than "***GESS* tabs** example `<tablenumber>`".

1: disable `VARTITLE`
Whether a `VARTITLE` is printed or not is one of the possible options of `TABLEFORMAT`. `TABLEFORMAT`s are presets like `TABLETYPE` etc; once they are written into the script they are valid until a new preset appears. Thus `TAB.INC` is amended **before** the `TABLE` statement:

```
TABLEFORMAT = +NOVARTITLEBOX;
```

2: define question text
Always those texts are to be printed that belong to the questions tabulated on the Y-Axis. The table has two possible positions for them, namely `TOPTEXT` and `BOTTOMTEXT`. The text is usually above the actual table and below the `TABLETITLE`. Therefore we amend `TAB.INC`:

```
CITEALLVARS = TOPTEXT YVALID;
```

Now we change the table title:

```
TABLETITLE = "GESS tabs Example Nr. _";
```

`TAB.INC` now looks like this:

```
TABLETITLE = "GESS tabs Example Nr. _";
CITEALLVARS = TOPTEXT YVALID;
```

[1] The question text can also be added directly to the table but then we don't always want to re-type or copy as it has already been written out in `EXAMPLE.INC` (e.g. for CATI or CAPI surveys). Quoting avoids redundancy and thus avoids errors.

```
TABLEFORMAT = NOVARTITLEBOX;

TABLE = #k BY bekannt;
```

This is the result:

GESS tabs Beispiel Nr. 4

Welche dieser Mobiltelefone kennen Sie?

Col %	Total	Geschlecht		Alter		
		Männlich	Weiblich	18-29 Jahre	30-49 Jahre	50-64 Jahre
N	482	255	227	162	157	163
Alcatel	56.2 %	54.1 %	58.6 %	51.9 %	59.2 %	57.7 %
Nokia	60.6 %	58.0 %	63.4 %	59.3 %	64.3 %	58.3 %
Motorola	60.4 %	61.2 %	59.5 %	58.0 %	65.6 %	57.7 %
Panasonic	59.1 %	55.3 %	63.4 %	61.1 %	57.3 %	58.9 %
Sagem	63.7 %	64.3 %	63.0 %	60.5 %	64.3 %	66.3 %
Samsung	63.5 %	59.2 %	68.3 %	59.9 %	65.6 %	65.0 %
Siemens	57.5 %	54.1 %	61.2 %	54.3 %	65.0 %	53.4 %
Sony-Ericsson	60.6 %	61.6 %	59.5 %	58.6 %	56.7 %	66.3 %
keine davon	9.8 %	11.8 %	7.5 %	14.8 %	8.3 %	6.1 %

GESS mbH *

GESS

It would be clearer if the percentages in the total column were presented in descending order which would reflect the basic relevant absolute frequency. This can be achieved using:

```
TABLE = #k BY bekannt SORT ABSOLUTE DESCEND;
```

GESS tabs Beispiel Nr. 5

Welche dieser Mobiltelefone kennen Sie?

		Geschlecht		Alter		
Col %	Total	Männlich	Weiblich	18-29 Jahre	30-49 Jahre	50-64 Jahre
N	482	255	227	162	157	163
Sagem	63.7 %	64.3 %	63.0 %	60.5 %	64.3 %	66.3 %
Samsung	63.5 %	59.2 %	68.3 %	59.9 %	65.6 %	65.0 %
Nokia	60.6 %	58.0 %	63.4 %	59.3 %	64.3 %	58.3 %
Sony-Ericsson	60.6 %	61.6 %	59.5 %	58.6 %	56.7 %	66.3 %
Motorola	60.4 %	61.2 %	59.5 %	58.0 %	65.6 %	57.7 %
Panasonic	59.1 %	55.3 %	63.4 %	61.1 %	57.3 %	58.9 %
Siemens	57.5 %	54.1 %	61.2 %	54.3 %	65.0 %	53.4 %
Alcatel	56.2 %	54.1 %	58.6 %	51.9 %	59.2 %	57.7 %
keine davon	9.8 %	11.8 %	7.5 %	14.8 %	8.3 %	6.1 %

GESS mbH *

GESS

Two further tables have been added:

```
TABLE = #k BY gekauft SORT ABSOLUTE DESCEND;
TABLE = #k BY item_1;
```

GESS tabs Beispiel Nr. 6

Welche dieser Mobiltelefone haben Sie sich schon einmal gekauft?

		Geschlecht		Alter		
Col %	Total	Männlich	Weiblich	18-29 Jahre	30-49 Jahre	50-64 Jahre
N	482	255	227	162	157	163
Motorola	30.5 %	29.8 %	31.3 %	30.9 %	32.5 %	28.2 %
Panasonic	30.3 %	25.9 %	35.2 %	32.1 %	29.3 %	29.4 %
Sagem	29.5 %	31.8 %	26.9 %	26.5 %	33.8 %	28.2 %
Nokia	29.3 %	28.2 %	30.4 %	29.6 %	32.5 %	25.8 %
Sony-Ericsson	28.8 %	29.8 %	27.8 %	27.2 %	25.5 %	33.7 %
Samsung	28.8 %	26.3 %	31.7 %	27.8 %	30.6 %	28.2 %
Alcatel	28.2 %	28.6 %	27.8 %	23.5 %	31.8 %	29.4 %
Siemens	27.0 %	23.1 %	31.3 %	26.5 %	28.7 %	25.8 %
keine davon	33.2 %	36.1 %	30.0 %	35.2 %	33.1 %	31.3 %

GESS mbH *

GESS

Regarding table 6: the category "none of the above" is at the end of the column although its value is not the lowest. This is defined by the label characteristic of the label text "BOTTOM". (See the variable definition printed above.)

GESS tabs Beispiel Nr. 7

Bitte stufen Sie dieses Item ein:
Die Farbe ist mir wichtig

		Geschlecht		Alter		
Col %	Total	Männlich	Weiblich	18-29 Jahre	30-49 Jahre	50-64 Jahre
N	482	255	227	162	157	163
stimme ganz und gar nicht zu	20.3 %	22.4 %	18.1 %	21.0 %	22.3 %	17.8 %
stimme eher nicht zu	19.7 %	18.0 %	21.6 %	15.4 %	18.5 %	25.2 %
weder/noch	22.2 %	24.3 %	19.8 %	29.0 %	22.3 %	15.3 %
stimme eher zu	18.0 %	16.9 %	19.4 %	16.0 %	17.2 %	20.9 %
stimme ganz und gar zu	19.7 %	18.4 %	21.1 %	18.5 %	19.7 %	20.9 %

GESS mbH

GESS

Looking at item 1 of the table the idea of also having the mean as well as the frequency presents itself. This is achieved by including a MEAN clause in the TABLE statement. All CELLELEMENTS can be additionally presented as simple rows or columns. Means and similar measured values such as variance or median all need an additional variable as an argument:

```
TABLE = #k BY item_1 mean :description "Mittelwert" ( item_1 );
```

GESS tabs Beispiel Nr. 8

Bitte stufen Sie dieses Item ein:
Die Farbe ist mir wichtig

		Geschlecht		Alter		
Col %	Total	Männlich	Weiblich	18-29 Jahre	30-49 Jahre	50-64 Jahre
N	482	255	227	162	157	163
stimme ganz und gar nicht zu	20.3 %	22.4 %	18.1 %	21.0 %	22.3 %	17.8 %
stimme eher nicht zu	19.7 %	18.0 %	21.6 %	15.4 %	18.5 %	25.2 %
weder/noch	22.2 %	24.3 %	19.8 %	29.0 %	22.3 %	15.3 %
stimme eher zu	18.0 %	16.9 %	19.4 %	16.0 %	17.2 %	20.9 %
stimme ganz und gar zu	19.7 %	18.4 %	21.1 %	18.5 %	19.7 %	20.9 %
Mittelwert	3,0	2,9	3,0	3,0	2,9	3,0

GESS mbH

GESS

In the table up until now each cell has only contained one value: column percentage. Several values can be presented in the individual cells of a table simultaneously. ***GESS* tabs** supports up to seven

CELLELEMENTS per cell. The CELLELEMENTS clause merely needs to be expanded and for this example the cell mode ABSOLUTE is used:

```
CELLELEMENTS = COLUMNPERCENT ABSOLUTE;
TABLE = #k BY item_1 mean :description "Mittelwert" ( item_1 );
```

GESS tabs Beispiel Nr. 9

Bitte stufen Sie dieses Item ein:
Die Farbe ist mir wichtig

Col % Abs.	Total	Geschlecht		Alter		
		Männlich	Weiblich	18-29 Jahre	30-49 Jahre	50-64 Jahre
N	482	255	227	162	157	163
stimme ganz und gar nicht zu	20.3 % 98	22.4 % 57	18.1 % 41	21.0 % 34	22.3 % 35	17.8 % 29
stimme eher nicht zu	19.7 % 95	18.0 % 46	21.6 % 49	15.4 % 25	18.5 % 29	25.2 % 41
weder/noch	22.2 % 107	24.3 % 62	19.8 % 45	29.0 % 47	22.3 % 35	15.3 % 25
stimme eher zu	18.0 % 87	16.9 % 43	19.4 % 44	16.0 % 26	17.2 % 27	20.9 % 34
stimme ganz und gar zu	19.7 % 95	18.4 % 47	21.1 % 48	18.5 % 30	19.7 % 31	20.9 % 34
Mittelwert	3,0	2,9	3,0	3,0	2,9	3,0

GESS mbH

GESS

TABLE can also be expanded "downwards"; e.g. 2 items can be presented on one page with their relevant means. It is however perhaps clearer not to present all question texts in the TOPTEXT but rather to define the relevant characteristics in the VARTITLE.

```
CITEALLVARS = no;
TABLEFORMAT = -NOVARTITLEBOX;
CELLELEMENTS = COLUMNPERCENT;
table = #k by
item_1 mean :description "Mittelwert" ( item_1 )
item_2 mean :description "Mittelwert" ( item_2 )
;
```

GESS tabs Beispiel Nr. 10

Col %	Total	Geschlecht		Alter		
		Männlich	Weiblich	18-29 Jahre	30-49 Jahre	50-64 Jahre
N	482	255	227	162	157	163
Die Farbe ist mir wichtig						
stimme ganz und gar nicht zu	20.3 %	22.4 %	18.1 %	21.0 %	22.3 %	17.8 %
stimme eher nicht zu	19.7 %	18.0 %	21.6 %	15.4 %	18.5 %	25.2 %
weder/noch	22.2 %	24.3 %	19.8 %	29.0 %	22.3 %	15.3 %
stimme eher zu	18.0 %	16.9 %	19.4 %	16.0 %	17.2 %	20.9 %
stimme ganz und gar zu	19.7 %	18.4 %	21.1 %	18.5 %	19.7 %	20.9 %
Mittelwert	3,0	2,9	3,0	3,0	2,9	3,0
Eine gute Sprachqualität ist mir wichtig						
stimme ganz und gar nicht zu	23.7 %	24.3 %	22.9 %	25.9 %	23.6 %	21.5 %
stimme eher nicht zu	16.8 %	15.7 %	18.1 %	19.8 %	17.2 %	13.5 %
weder/noch	20.3 %	18.8 %	22.0 %	19.8 %	17.2 %	23.9 %
stimme eher zu	19.3 %	18.4 %	20.3 %	18.5 %	21.0 %	18.4 %
stimme ganz und gar zu	19.9 %	22.7 %	16.7 %	16.0 %	21.0 %	22.7 %
Mittelwert	3,0	3,0	2,9	2,8	3,0	3,1

GESS mbH

GESS

Often the stub contains columns which are first generated from a combination of several questions or variables. Such combinations are most elegantly achieved using the `GROUPS` statement:

```
GROUPS Männer_k =
|  "18-29 Jahre" : alter EQ 1 AND geschl EQ 1
|  "30-49 Jahre" : alter EQ 2 AND geschl EQ 1
|  "50-64 Jahre" : alter EQ 3 AND geschl EQ 1
;
VARTITLE = "Männer";

GROUPS Frauen_k =
|  "18-29 Jahre" : alter EQ 1 AND geschl EQ 2
|  "30-49 Jahre" : alter EQ 2 AND geschl EQ 2
|  "50-64 Jahre" : alter EQ 3 AND geschl EQ 2
;
VARTITLE = "Frauen";

#expand #k2 Männer_k Frauen_k
```

The `GROUPS` statement used above is the clearest way of producing a new `DICHOQ`. The main advantage is that the text and the relevant requirements can be seen at a glance.

```
TABLE = #k BY bekannt SORT ABSOLUTE DESCEND;
```

GESS tabs Beispiel Nr. 11

Welche dieser Mobiltelefone kennen Sie?

Col %	Total	Männer			Frauen		
		18-29 Jahre	30-49 Jahre	50-64 Jahre	18-29 Jahre	30-49 Jahre	50-64 Jahre
N	482	85	80	90	77	77	73
Sagem	63.7 %	60.0 %	62.5 %	70.0 %	61.0 %	66.2 %	61.6 %
Samsung	63.5 %	56.5 %	63.8 %	57.8 %	63.6 %	67.5 %	74.0 %
Nokia	60.6 %	52.9 %	67.5 %	54.4 %	66.2 %	61.0 %	63.0 %
Sony-Ericsson	60.6 %	52.9 %	56.3 %	74.4 %	64.9 %	57.1 %	56.2 %
Motorola	60.4 %	56.5 %	66.3 %	61.1 %	59.7 %	64.9 %	53.4 %
Panasonic	59.1 %	52.9 %	50.0 %	62.2 %	70.1 %	64.9 %	54.8 %
Siemens	57.5 %	49.4 %	58.8 %	54.4 %	59.7 %	71.4 %	52.1 %
Alcatel	56.2 %	42.4 %	58.8 %	61.1 %	62.3 %	59.7 %	53.4 %
keine davon	9.8 %	20.0 %	11.3 %	4.4 %	9.1 %	5.2 %	8.2 %

GESS mbH

But the client wants something else! Would it be possible to present the five items not next to each other? Of course it would! But it is a little more complicated as this is no longer a classical cross table. One way to imagine such a table is addition; initially the first column is filled with the first variable, then the second column with the second variable, etc. and the mechanism of table addition (`TABLE ADD`) was already presented in the introduction.

Firstly we need a stub which contains the five items. As this is called a dummy variable in program speak, we shall call it `dummy5`.

```
COMPUTE dummy5 = 1;
VARTITLE dummy5 = "5 Items nebeneinander";
LABELS dummy5 =
1  "Farbe wichtig"
2  "Sprach~qualität wichtig"
3  "Gute Ausstat~tung wichtig"
4  "Marke bevorzugt"
5  "Marke abgelehnt"
;
#expand #k3 dummy5
```

Then we can write:

```
tableformat = +novartitlebox;
table     = #k3 by item_1;
table add = 2   by item_2;
table add = 3   by item_3;
table add = 4   by item_4;
table add = 5   by item_5;
```

The table statement now consists of 5 `TABLE` statements; one "normal" and then four `TABLE ADD` statements. In the `TABLE ADD` rows a value for the "suitable" label can be substituted for the variables. And this is the result:

GESS tabs Beispiel Nr. 12

Col %	Total	5 Items nebeneinander: Farbe wichtig	Sprach-qualität wichtig	Gute Ausstat-tung wichtig	Marke bevorzugt	Marke abgelehnt
N	2410	482	482	482	482	482
stimme ganz und gar nicht zu	19.9 %	20.3 %	23.7 %	17.0 %	19.1 %	19.3 %
stimme eher nicht zu	19.2 %	19.7 %	16.8 %	19.9 %	19.1 %	20.3 %
weder/noch	20.0 %	22.2 %	20.3 %	19.9 %	18.3 %	19.3 %
stimme eher zu	19.9 %	18.0 %	19.3 %	21.2 %	21.4 %	19.5 %
stimme ganz und gar zu	21.1 %	19.7 %	19.9 %	22.0 %	22.2 %	21.6 %

GESS mbH *

GESS

Examining the table reveals that it is not always wise to have a total column; it isn't technically incorrect but can be confusing. The `TOTALCOLUMN` as a `FRAMEELEMENT` is disabled and only `ABSROW` is used.

Additionally we would like to have the mean back which involves requesting the `MEAN` for all five items:

```
frameelements = absrow ;
table     = #k3 by item_1 mean : description "Mittelwert" ( item_1 );
table add = 2   by item_2 mean : description "Mittelwert" ( item_2 );
table add = 3   by item_3 mean : description "Mittelwert" ( item_3 );
table add = 4   by item_4 mean : description "Mittelwert" ( item_4 );
table add = 5   by item_5 mean : description "Mittelwert" ( item_5 );
```

And this is the resulting table:

GESS tabs Beispiel Nr. 13

Col %	5 Items nebeneinander: Farbe wichtig	Sprach-qualität wichtig	Gute Ausstat-tung wichtig	Marke bevorzugt	Marke abgelehnt
N	482	482	482	482	482
stimme ganz und gar nicht zu	20.3 %	23.7 %	17.0 %	19.1 %	19.3 %
stimme eher nicht zu	19.7 %	16.8 %	19.9 %	19.1 %	20.3 %
weder/noch	22.2 %	20.3 %	19.9 %	18.3 %	19.3 %
stimme eher zu	18.0 %	19.3 %	21.2 %	21.4 %	19.5 %
stimme ganz und gar zu	19.7 %	19.9 %	22.0 %	22.2 %	21.6 %
Mittelwert	3,0	3,0	3,1	3,1	3,0

GESS mbH *

GESS

The question can now be posed as to whether the mean between the items is significant, i.e. whether they vary more than would be expected from the sample. This involves replacing the simple `CELLELEMENT MEAN` with the more complex `CELLELEMENT MEANTEST`. Additionally the `TABLEFORMAT AUTOSIGNCHAR` is brought into play. This causes the letters in the label boxes in the

stub to be printed which serves to cross-reference which means vary significantly. Furthermore we require a test at a low level: the `SIGNIFLEVEL` is to be `LOWSIGNIFICANCE`. And with a small trick we shall achieve the documentation of the test level: we shall include an empty bottom text. If a table has a bottom text the descriptive text of the chosen significant level is included in it. This bottom text has to be written **before** the `TABLE ADD` statements so that it is allocated to the printed table; the remaining `TABLE ADD` statements only cause so-called "counting shadows".

```
TABLEFORMAT = +AUTOSIGNCHAR;
SIGNIFLEVEL = LOWSIGNIFICANCE;

table     = #k3 by item_1 meantest : description "Mittelwert" (
item_1 );
bottomtext = "";
table add = 2   by item_2 meantest : description "Mittelwert" (
item_2 );
table add = 3   by item_3 meantest : description "Mittelwert" (
item_3 );
table add = 4   by item_4 meantest : description "Mittelwert" (
item_4 );
table add = 5   by item_5 meantest : description "Mittelwert" (
item_5 );
```

And this is the result: As long as we can live with the relatively high probable error of 10% the difference between `Item_2` and `Item_3` can be considered significant.

GESS tabs Beispiel Nr. 14

	5 Items nebeneinander				
	Farbe wichtig	Sprach-qualität wichtig	Gute Ausstat-tung wichtig	Marke bevorzugt	Marke abgelehnt
Col %	A	B	C	D	E
N	482	482	482	482	482
stimme ganz und gar nicht zu	20.3 %	23.7 %	17.0 %	19.1 %	19.3 %
stimme eher nicht zu	19.7 %	16.8 %	19.9 %	19.1 %	20.3 %
weder/noch	22.2 %	20.3 %	19.9 %	18.3 %	19.3 %
stimme eher zu	18.0 %	19.3 %	21.2 %	21.4 %	19.5 %
stimme ganz und gar zu	19.7 %	19.9 %	22.0 %	22.2 %	21.6 %
Mittelwert	3,0	3,0 c	3,1 b	3,1	3,0

Signif. 5% (ABC...) / 10% (abc...)

GESS mbH *

GESS

As you can see here in the standard case for `MEANTEST` the indication of significance of the difference between column B and column C is shown twice: in column B there is a small C and vice versa. This can be modified using the `SHOWSIGNIF` statement.

```
SHOWSIGNIF MEAN = GREATER;
```

This causes the printing of only the larger mean of a column reference:

GESS tabs Beispiel Nr. 15

	5 Items nebeneinander				
	Farbe wichtig	Sprach-qualität wichtig	Gute Ausstat-tung wichtig	Marke bevorzugt	Marke abgelehnt
Col %	A	B	C	D	E
N	482	482	482	482	482
stimme ganz und gar nicht zu	20.3 %	23.7 %	17.0 %	19.1 %	19.3 %
stimme eher nicht zu	19.7 %	16.8 %	19.9 %	19.1 %	20.3 %
weder/noch	22.2 %	20.3 %	19.9 %	18.3 %	19.3 %
stimme eher zu	18.0 %	19.3 %	21.2 %	21.4 %	19.5 %
stimme ganz und gar zu	19.7 %	19.9 %	22.0 %	22.2 %	21.6 %
Mittelwert	3,0	3,0	3,1 b	3,1	3,0

Signif. 5% (ABC...) / 10% (abc...)

GESS mbH *

GESS

A suggestion: it could make things clearer if the software coloured the areas where there are significant differences between the means. There is the general possibility of defining colours for individual value areas of `CELLELEMENTS`. The significance related `CELLELEMENTS` delivers a synthetic value of 99999 if a test causes the printing of a significant reference. This value can be used in order to colour significantly varying cells:

```
RGB = YES;
COLOR FOREGROAND =
| MEANTEST RANGE 99998.9 99999.1 : 1 0 0  // rot!
;
```

The `COLOR` statement can be used to colour the foreground or the background. In this case the text, the `FOREGROUND`, has been changed to bright red. And the result looks like this:

GESS tabs Beispiel Nr. 16

	5 Items nebeneinander				
	Farbe wichtig	Sprach-qualität wichtig	Gute Ausstat-tung wichtig	Marke bevorzugt	Marke abgelehnt
Col %	A	B	C	D	E
N	482	482	482	482	482
stimme ganz und gar nicht zu	20.3 %	23.7 %	17.0 %	19.1 %	19.3 %
stimme eher nicht zu	19.7 %	16.8 %	19.9 %	19.1 %	20.3 %
weder/noch	22.2 %	20.3 %	19.9 %	18.3 %	19.3 %
stimme eher zu	18.0 %	19.3 %	21.2 %	21.4 %	19.5 %
stimme ganz und gar zu	19.7 %	19.9 %	22.0 %	22.2 %	21.6 %
Mittelwert	3,0	3,0	3,1 b	3,1	3,0

Signif. 5% (ABC...) / 10% (abc...)

GESS mbH *

GESS

Not only can the means of the scale be tested against each other but also the column percentages of the individual cells of the matrix can be tested for significant difference. This merely involves including the relevant `CELLELEMENT`:

```
CELLELEMENTS = COLUMNPERCENT COLCHIQU;
```

And as we would like to have a colour reference here also we need to expand the `COLOR FOREGROUND` statement:

```
RGB = YES;
COLOR FOREGROUND =
| MEANTEST RANGE 99998.9 99999.1 : 1 0 0
| COLCHIQU RANGE 99998.9 99999.1 : 1 0 0
;
```

The improved table looks like this:

GESS tabs Beispiel Nr. 17

	5 Items nebeneinander				
	Farbe wichtig	Sprach-qualität wichtig	Gute Ausstat-tung wichtig	Marke bevorzugt	Marke abgelehnt
Col % ColSig	A	B	C	D	E
N	482	482	482	482	482
stimme ganz und gar nicht zu	20.3 %	23.7 % Cde	17.0 %	19.1 %	19.3 %
stimme eher nicht zu	19.7 %	16.8 %	19.9 %	19.1 %	20.3 %
weder/noch	22.2 %	20.3 %	19.9 %	18.3 %	19.3 %
stimme eher zu	18.0 %	19.3 %	21.2 %	21.4 %	19.5 %
stimme ganz und gar zu	19.7 %	19.9 %	22.0 %	22.2 %	21.6 %
Mittelwert	3,0	3,0	3,1 b	3,1	3,0

Signif. 5% (ABC...) / 10% (abc...)

GESS mbH *

If it should be required that the five adjacent items be presented separately for men and women it is necessary to basically present the same information twice, filtered according to sex. It is simplest to select the cases for each table using `TABSELECT` or `TABLEFILTER`. This always concerns the whole table. Moreover there is also the possibility of using a filter clause within a `TABLE` statement which only concerns individual table elements.

In connection to every table element requested

```
FILTER <Bedingung> |
```

can define an internal selection.

As an example, a table as above has been requested but it is to contain two parts, one for men and one for women. Firstly an `#expand` is required for the stub which contains two dummy variables for the title "men" and "women". These dummy variables will also receive the relevant filter:

```
compute dummy5_2 dummy5_3 = 1;
copylabels  dummy5_2 dummy5_3 = dummy5;

filter dummy5_2 = geschl eq 1;
vartitle dummy5_2 = 'Männer';

filter dummy5_3 = geschl eq 2;
vartitle dummy5_3 = 'Frauen';

#expand #k4  dummy5_2 dummy5_3
```

An alternative to producing the requested table would be to create similar dummy variables for `Item_2 - Item_5`. It is however simpler to use the above possibility which filters the table elements directly in the `TABLE` statement. The command order is as below:

```
CELLELEMENTS = COLUMNPERCENT COLCHIQU;

TABLE     = #k4 BY item_1 MEANTEST : DESCRIPTION "Mittelwert" (
item_1 );
bottomtext = "";

TABLE ADD = 2 FILTER geschl EQ 1 | 2 FILTER geschl EQ 2 | BY item_2
MEANTEST : DESCRIPTION "Mittelwert" ( item_2 );

TABLE ADD = 3 FILTER geschl EQ 1 | 3 FILTER geschl EQ 2 | BY item_3
MEANTEST : DESCRIPTION "Mittelwert" ( item_3 );

TABLE ADD = 4 FILTER geschl EQ 1 | 4 FILTER geschl EQ 2 | BY item_4
MEANTEST : DESCRIPTION "Mittelwert" ( item_4 );

TABLE ADD = 5 FILTER geschl EQ 1 | 5 FILTER geschl EQ 2 | BY item_5
MEANTEST : DESCRIPTION "Mittelwert" ( item_5 );
```

And here is the requested table:

GESS tabs Beispiel Nr. 18

	Männer					Frauen				
	Farbe wichtig	Sprach-qualität wichtig	Gute Ausstat-tung wichtig	Marke bevorzugt	Marke abgelehnt	Farbe wichtig	Sprach-qualität wichtig	Gute Ausstat-tung wichtig	Marke bevorzugt	Marke abgelehnt
Col % ColSig	A	B	C	D	E	A	B	C	D	E
N	255	255	255	255	255	227	227	227	227	227
stimme ganz und gar nicht zu	22.4 % CD	24.3 % CDe	14.9 %	14.9 %	18.0 %	18.1 %	22.9 %	19.4 %	23.8 %	20.7 %
stimme eher nicht zu	18.0 %	15.7 %	23.1 % Bd	17.3 %	21.2 %	21.6 %	18.1 %	16.3 %	21.1 %	19.4 %
weder/noch	24.3 % Ce	18.8 %	17.3 %	20.8 %	17.6 %	19.8 %	22.0 % d	22.9 % D	15.4 %	21.1 %
stimme eher zu	16.9 %	18.4 %	20.4 %	22.4 %	21.2 %	19.4 %	20.3 %	22.0 %	20.3 %	17.6 %
stimme ganz und gar zu	18.4 %	22.7 %	24.3 %	24.7 % a	22.0 %	21.1 %	16.7 %	19.4 %	19.4 %	21.1 %
Mittelwert	2,9	3,0	3,2 A	3,2 AB	3,1	3,0	2,9	3,1	2,9	3,0

Signif. 5% (ABC...) / 10% (abc...)

GESS mbH

Perhaps the client would like men and women to be presented above each other instead of next to each other. One way would be to create variables filtered by sex for each of the five items. This is achieved with the following macros:

```
#macro #m( &1 )
COMPUTE m_item_&1 = item_&1;
FILTER m_item_&1 = geschl eq 1;
COPYLABELS m_item_&1 = item_&1;
VARTITLE m_item_&1 = 'Männer';
#endmacro

#macro #f( &1 )
COMPUTE f_item_&1 = item_&1;
FILTER f_item_&1 = geschl eq 2;
```

```
COPYLABELS f_item_&1 = item_&1;
VARTITLE f_item_&1 = 'Frauen';
#endmacro
```

Apart from the "normal" request for macros there is also the command `#domacro` which allows us to create the necessary ten variables most economically:

```
#domacro ( m 1:5 )
#domacro ( f 1:5 )
```

After the request for the macros there are additionally ten variables; `m_item_1 - m_ item _5` for men and the five variables for women. So as not to have to repeatedly write the description clause the `TABLEFORMAT MEANDESCRIPTION` and the `MEANTEST` are used for the descriptive text "mean". In order for the filter to be used on the filtered variable there is a `USECASES` clause which only effects the cases which are valid for the Y-axis tabled variables (= not filtered). The `TABLEFORMAT -NOVARTITLEBOX` ensures that the `VARTITLE` for men and women appears on the left above the labels. The table commands are thus as follows:

```
FRAMEELEMENTS =  ;
USECASES = YVALID;
TABLEFORMAT = +MEANDESCRIPTION -NOVARTITLEBOX;
DESCRIPTION MEANTEST = Mittelwert;
CELLELEMENTS = COLUMNPERCENT COLCHIQU;
TABLE      = #k3  BY m_item_1 MEANTEST ( m_item_1 ) f_item_1 MEANTEST
( f_item_1 );
BOTTOMTEXT = "";
TABLE ADD  =   2  BY m_item_2 MEANTEST ( m_item_2 ) f_item_2 MEANTEST
( f_item_2 );
TABLE ADD  =   3  BY m_item_3 MEANTEST ( m_item_3 ) f_item_3 MEANTEST
( f_item_3 );
TABLE ADD  =   4  BY m_item_4 MEANTEST ( m_item_4 ) f_item_4 MEANTEST
( f_item_4 );
TABLE ADD  =   5  BY m_item_5 MEANTEST ( m_item_5 ) f_item_5 MEANTEST
( f_item_5 );
```

GESS tabs Beispiel Nr. 19

Col % ColSig	5 Items nebeneinander Farbe wichtig A	Sprach-qualität wichtig B	Gute Ausstat-tung wichtig C	Marke bevorzugt D	Marke abgelehnt E
Männer					
stimme ganz und gar nicht zu	22.4 % CD	24.3 % CDe	14.9 %	14.9 %	18.0 %
stimme eher nicht zu	18.0 %	15.7 %	23.1 % Bd	17.3 %	21.2 %
weder/noch	24.3 % Ce	18.8 %	17.3 %	20.8 %	17.6 %
stimme eher zu	16.9 %	18.4 %	20.4 %	22.4 %	21.2 %
stimme ganz und gar zu	18.4 %	22.7 %	24.3 %	24.7 % a	22.0 %
Mittelwert	2,9	3,0	3,2 A	3,2 AB	3,1
Frauen					
stimme ganz und gar nicht zu	18.1 %	22.9 %	19.4 %	23.8 %	20.7 %
stimme eher nicht zu	21.6 %	18.1 %	16.3 %	21.1 %	19.4 %
weder/noch	19.8 %	22.0 % d	22.9 % D	15.4 %	21.1 %
stimme eher zu	19.4 %	20.3 %	22.0 %	20.3 %	17.6 %
stimme ganz und gar zu	21.1 %	16.7 %	19.4 %	19.4 %	21.1 %
Mittelwert	3,0	2,9	3,1	2,9	3,0

Signif. 5% (ABC...) / 10% (abc...)

GESS mbH *

GESS

The Syntax of the TABLE statement

```
TABLE [ taboptions ] = <parts> BY <parts>;

taboptions ::=
[
ADD
NAME <tablename>
TITLE <tabletitle>
CELLELEMENTS ( <cellelements> )
FRAMEELEMENTS ( <frameelements> )
TABLEFORMATS ( <tableformats> )
CONTENTKEY <contentkey>
]

parts ::= part { part }*n

part ::= content [ filter ] [ option ]

content ::=
[
<constant> |
<varname> |
<cellelement> ( <varname> [ <varname> ) |
<cellelement> ( <varname> [ <varname> ] BY <varname> )
]

filter ::= FILTER <bedingung> |

option ::= SORT sortcontent [ sortpane ] [ cut ]

sortcontent ::= [ DESCEND ] sorttype

sortpane ::= PANE <value> CODE <value>

cut ::=
[
TOP <value > [ SLICE <value> ] |
BOTTOM <value> |
EXTREME <value> |
SLICE <value> |
LSLICE <value> |
RANGE <value> <value>
]

sorttype ::= [ POSITION | ALPHA | CODE | Cellelement ]
```

The control of the table cell contents cells occurs fundamentally using the key word CELLELEMENTS. Using the key word FRAMEELEMENTS the table frame is defined. Additionally there is the key word TABLETYPE: this can be used to define a table in short form, with cells consisting of just one element, and the software deduces the frame elements. Thus a COLUMNPERCENT table has as a frame a total column and an absolute row. TABLETYPE and CELLELEMENTS use the same key words as arguments; CELLELEMENTS can use several arguments (the limit is at around seven).

Over the years a large number of CELLELEMENTS have collected at ***GESS* tabs**. There is a pivotal difference: a series of CELLELEMENTS e.g. COLUMNPERCENT or ABSOLUTE are complete in

themselves. Others e.g. MEAN or VARIANCE are only complete once a variable has been named to which it refers. And there are also some CELLELEMENTS which need more than one variable.

CELLELEMENTS are often requested for all the table cells, e.g.:

```
CELLELEMENTS = PCNTL1 MEDIAN PCNTL2;
FRAMEELEMENTS = TOTALROW TOTALCOLUMN;
TABLE = Wohnlage BY Wohnungsgröße;
```

or

```
CELLELEMENTS = ABSOLUTE MEAN( Einkommen );
FRAMEELEMENTS = TOTALROW TOTALCOLUMN;
TABLE = Status BY Geschlecht;
```

All key words for CELLELEMENTS can also be requested for individual table columns or rows, e.g.:

```
TABLE = #k BY skala_1 MEAN( skala1 );
```

Within the TABLE statement the interpreter uses the brackets to recognise that independent columns or rows are being referred to. Independent means that the CELLELEMENTS definition that normally controls the cell contents does not apply to these columns or rows.

A "local" description can also be defined using additional columns or rows by setting a colon in order to mark means etc., e.g.:

```
TABLE = Kopf BY MEAN :DESCRIPTION "Mittelwert" ( Einkommen );
```

The key word USEVARTITLE can also be used to request a "local" description of a VARTITLE:

```
TABLE = #k BY MEAN :USEVARTITLE x1 ( v1 );
```

A "local format" can also be defined in the same way:

```
TABLE = Kopf BY MEAN :FORMAT "#.##" ( Einkommen );
```

As mentioned above the CELLELEMENTS vary in that some expect one or more variables as arguments. These are measures such as MEAN or SUM or MEDIAN. Other CELLELEMENTS such as COLUMNPERCENT do not require these arguments; the content to be displayed is calculated from the variables that define the tables i.e. the variables in the stub and their characteristics.

These CELLELEMENTS can request additional rows or columns, e.g.:

```
TABLE = #k BY var1 PROJECTION():
```

All these spreads can be filtered. Additionally a variable name can be stated in brackets. During calculation the MISSING status of these variables is evaluated: if these variables are MISSING in a case it will not be counted. NB: only the "genuine" MISSING are referred to here; a filter on the relevant variable does not count as MISSING. Using COMPUTE (e.g. COMPUTE v = v;) can generate a genuine MISSING if necessary.

The table content is controlled by the content of the individual table cells and the additional frame elements. The contents of the individual cells are referred to as cellelements; the key word in ***GESS* tabs** is "CELLELEMENTS". The elements of the table frame, i.e. the absolute columns and rows or total columns and rows are referred to as frame elements; the key word here is "FRAMEELEMENTS".

The USECASES command is used to influence the inclusion rule for calculating tables and thus the percentage basis. The options are described in detail below.

CELLELEMENTS like TABLETYPE sets a preset for all the following tables which remains true until a new CELLELEMENTS or TABLETYPE command occurs. The calculation of further variables can be requested in CELLELEMENTS. All commands have to refer to the same additional variable apart from in the exception described below. Thus e.g.:

```
CELLELEMENTS = ABSOLUTE MEAN( v1 ) SUM( v1 ) STDDEV( v1 );
```

is allowed whereas

```
CELLELEMENTS = MEAN( v1 ) SUM( v2 );
```

is not.

There are two additional CELLELEMENTs: if two means or two sums are to appear in one row the mean or the sum can be requested via the second additional variable using the key word SECONDMEAN or SECONDSUM.
Example:

```
CELLELEMENTS = MEAN( v1 ) SECONDMEAN( v2 );
```

In exactly this case reference to another additional variable is permitted.

***GESS* tabs** recognises the following CELLELEMENTS:

ABSCOLPERCENT	as COLUMNPERCENT, additionally prints the absolute frequency
ABSMEAN	as MEAN, additionally prints the absolute frequency
ABSMEANSUM (Var **)**	tally, sum and mean of a row
ABSOLUTE	absolute frequency value: sum of weights
ABSROWPERCENT	as ROWPERCENT, additionally prints absolute frequency
CHIQU	prints the chi-square test per row. The chi-square test values the variance of the empirical distribution in each row on the basis of the expected values calculated by the marginal distribution
COLCHIQU	4-field chi-square test of all cells in a table row against each other; marked with the letters from INDEXCHARS.
COLDEPTTEST (Var)	prints column-based significant letters for the dependent t-test. This test only makes noticeable sense if the cells have been created by addition (TABLE ADD...) otherwise it MUST go wrong. As a reminder: the difference between the scale values is calculated for each case and the mean of this difference is tested against zero. This test only makes sense between different variables.
COLPCTBENCHMARK	compares column percentages with externally defined bench mark values. See extra explanation.
COLPERCANDSIGN	COLUMNPERCENT and significance letters in one CELLELEMENT.
COLPERCENTDELTA	prints a table with delta values (in percent points) to the value of the total column.
COLPERCENTINDEX	prints a table with index values (100 corresponds to the value in the total column) to the column percent
COLPERCENTMEAN(Var)	column percent and mean in one row
COLPERCENTSUM(Var)	column percent and sum as a combined CELLELEMENT
COLPERCEQUAL	tests all column percentages in the column for equality; i.e. all variances from the equal distribution are considered significant. Here there is the possibility of producing many meaningless significances. Please use with care.

COLPERCSTDERR	standard error of the column percent value.
COLPERCT	calculates a t-test across the column percent values. In this t-test a correction on the basis of the weighting is carried out and the column overlaps are taken into consideration. TESTCOLUMNS are taken into account.
COLPERCZ	test per column of the differences in the column percentages with the z-test. The extended z-test with arc-sine-correction is used.
COLROWPERCENT	column and row percentages in one row
COLSUMPERCENT (Var);	printing of column percentages of the sum of a third variable e.g. the sum of the expenditure for a certain cause in certain areas etc.
COLUMNPERCENT	column percents
COLUMNRANGE	prints a table with the lower and upper edges of a confidence interval (5%) from the column percentages
CONFIDENCERANGE (Var)	prints the confidence interval of an additional variable. Two values in one row.
CUMULATIVE	percentaged and accumulated per row.
DELTAEXPECT	prints the difference between the empirical cell value and the expected cell value after marginal distribution.
DELTAPERCENT (Var, BasisVar)	sum of VAR and BASISVAR are calculated and the difference is percentaged on the BASISVAR.
DELTAPOINTS (Var, BasisVar)	sum of VAR and BASISVAR are calculated and the difference is percentaged on the number of valid cases.
DELTASUMPERCENT (Varfamily)	VARFAMILY must contain four single variables. These denote the relevant enumerator and denominator of a fraction. The sum across the enumerator and denominator is calculated and the difference between the quotients is shown as a percentage.
EXPECT	prints the expected cell value on the basis of marginal distribution.
GEOMETRICMEAN (Var)	geometric mean is the n-root of the product of all single values. The geometric mean is also only defined for positive figures.
HARMONICMEAN (Var)	special mean, the harmonic mean. The harmonic mean is the reverse value of the mean of the reverse value. The harmonic mean is only defined for positive figures. Is used in special cases, e.g. as the mean of speeds etc.
MEAN (Var)	mean
MEAN_PHYS (Var)	weighted mean and unweighted base
MEANCOLDEPT (Var)	mean as above additionally expanded to the printing of a t-test-significant level. This is a dependent t-test. This test makes no sense in any tables where it is differentiated between groups (see COLDEPTTEST). TESTCOLUMNS are taken into account.
MEANCOLINDEX(Var)	presentation of the mean, each on the mean of the total column normed to 100.
MEANCUT (Var)	mean MEANCUT cuts off extreme values at the lower and upper end of the distribution and calculates the mean on the basis of the remaining distribution per cell. If the extreme group cannot be created from complete cases a proportionate weighting is used. the size of the extreme cut is defined in percent points: BOTTOMCUT = <number>; (Voreinstellung 3%) TOPCUT = <number>; (Voreinstellung 3%)
MEANTEST (Var)	mean as above, additionally expanded to the printing of a t-test-significant level; tested against the other columns. As in ABSCOLPERCENT two text elements are presented in one row. The specification of a tab value for the relevant cells is

	recommended for Postscript-output (DATACELL etc). TESTCOLUMNS are taken into account.
MEANROWINDEX(Var)	presentation of the mean each on the mean of the total row normed to 100.
MEDIAN (Var)	median of a third variable in all cells. With median as with all percentiles ***GESS* tabs** will interpolate if it is requested with the TABLEFORMAT PERCENTILEINTERPOL command. MEDIAN and PCNTL1 - PCNTL4 can be combined within a row; first PCNTL1, then MEDIAN and lastly PCNTL2 are printed under each other.
MODE (Var)	mode
PCNTL1 (Var) **PCNTL2** (Var) **PCNTL3** (Var) **PCNTL4** (Var)	arbitrary percentile. For PCNTL1 the 1st quartile and for PCNTL2 the 3rd quartile are preset, i.e. 25 or 75% each of the cell distribution. With additional statements the limit and the text of the percentile analysis can be chosen individually, e.g.: PCNTL1 = 33.333% "1.Drittel"; PCNTL2 = 66.667% "2.Drittel"; It will then be interpolated if requested by TABLEFORMAT PERCENTILEINTERPOL.
PCNTRANGE(Var)	prints 1st and 2nd percentiles as a range in one row.
PERCENTILEDELTA (Var)	prints difference between the 1st and 2nd percentile.
PHYSCOLCHIQU	4-field chi-square test of all rows in a table row against each other; marking with a letter from INDEXCHARS. Also in a weighted table the unweighted case numbers are taken as a basis.
PHYSCOLDELTA	difference between weighted and unweighted column percentages.
PHYSCOLPERCENT	as COLUMNPERCENT, but on the basis of unweighted figures.
PHYSICALRECORDS	as ABSOLUTE, but the number of physically present cases is counted instead of the weights.
PHYSMEAN (Var)	unweighted mean.
PHYSMEANTEST(Var)	unweighted counterpart to MEANTEST. Mean and test on the basis of the physical frequency.
PHYSROWDELTA	difference between the weighted and unweighted row percentages.
PHYSROWPERCENT	row percentage on the basis of an unweighted count.
PHYSTTEST (Var)	index initial for t-test on the basis of unweighted values. (INDEXCHARS)
PROJCOLPERCENT	as COLUMNPERCENT, but additional printing of the projection.
PROJECTION	as ABSOLUTE, but the weights are multiplied with the PROJECTIONFACTOR. Here a random sample can be extrapolated to the basic population using the weighted distribution. The PROJECTIONFACTOR is set using PROJECTIONFACTOR = <Wert>; Presetting: 1.0.
PROJECTIONSUM (Var)	as SUM, but the totals are multiplied by PROJECTIONFACTOR.
ROWCHIQU	ROWCHIQU demands a 4-field chi-square test in the cells of every column against the relevant cells in all the other rows. The identification occurs analogue to COLCHIQU with alphabetical identification, A is the first row, B the second, etc. Individual orders can be defined with INDEXCHARS.
ROWMEANTEST (Var)	as MEANTEST, but the values in the rows are tested against each other.

ROWPERCENT	row percentages.
ROWPERCENTINDEX	table cells contain index values (100 = the value in the total row) to the row percentages.
ROWPERCEQUAL	tests all row percentages in the row for equality; i.e. all variances from the equal distribution are considered significant. Here there is the danger of producing many meaningless significances. Please use with care.
ROWPERCSTDERR	standard error of the row percentage values.
ROWPERCZ	significance test. ROWPERCZ is based on the z-test for percentage values. The expanded z-test with arc-sine correction is used here.
ROWSUMPERCENT (Var)	prints the row percentages of the sum of a three-way variable, e.g. the sum of expenditure for a certain cause in certain areas etc.
SECONDMEAN (Var)	2nd mean. If the mean of two different variables is to be printed in a row, the second variable must be requested using SECONDMEAN.
SECONDSUM (Var)	2nd Sum. If the sum of two different variables is to be printed in a row, the second variable must be requested using SECONDSUM.
STDDEV (Var)	standard deviation.
STDERR (Var)	standard error.
SUM (Var)	presentation of the sum of VAR.
SUMPERCENT (Var, BasisVar)	sum is calculated on the basis of VAR and BASISVAR. The sum of VAR is presented as the percentages quotient of the sum of BASISVAR.
SUMQUOTIENT (Var, BasisVar)	sum is calculated on the basis of VAR and BASISVAR. The sum of VAR is presented as the quotient of the sum of BASISVAR.
TOTALPERCENT	percentage of all cells on the basis of the table's total number of cases.
TOTALPERCSTDERR	standard error of the total percentage value.
TOTALSUMPERCENT (Var)	prints the percentage of the sum of a third variable on the basis of the total in the table.
TTEST (Var **)**	prints the result of the significance test. TESTCOLUMNS are taken into account; alphabetical identification can be set using INDEXCHARS.
VARIANCE (Var)	variance
VARIATION (Var)	variation coefficient
ZRANGE (Var)	prints the central area of a variable, mean, +/- scatter, * ZVALUE. This Factor can be arbitrarily chosen using ZVALUE, e.g.: ZVALUE = 1.0; Preset: ZVALUE = 0.967; (2/3 range of the mean).

As a result of internal memory structures there are some incompatibilities within cell contents. Thus HARMONICMEAN and GEOMETRICMEAN cannot be combined with other cell contents and MEDIAN is only compatible with frequency and percentage and not with other sums, means or scatters.

CELLSEQUENCE

If several CELLELEMENTS are required for a table the cell contents are printed underneath each other in a standard order. This standard order can be altered using the CELLSEQUENCE statement. CELLSEQUENCE defines a new order. All CELLELEMENTS which do not appear in CELLSEQUENCE are not printed.

CELLSET

The CELLELEMENTS statement can be used to combine several pieces of information in a single table cell in the parts of the table which span across both axes using LABELS. In summary tables additional summarised rows are often required where e.g. means are to be presented. Due to the syntax this is only one CELLELEMENT, if necessary this can be one that includes two values e.g. ABSMEAN or MEANTEST.

The CELLSET statement can be used to form individual "synthetic" summaries of CELLELEMENTS in additional rows that can be printed together.

e.g. an individual CELLSET can be defined, which calculates the mean of three different variables and the sum of a fourth variable:

```
CELLSET MySet( p1 p2 p3 p4 ) = mean( p1 ) mean( p2 ) mean( p3 ) sum (
p4 );
```

p1 - p4 here are formal parameters which are replaced by real variables using CELLSET MySet, e.g.

```
TABLE = #k BY CELLSET MySet( f1_1 f1_2 f1_3 f2 );
```

Due to internal technical reasons not all ***GESS* tabs** CELLELEMENTS can be used in a CELLSET statement.

The following can:

ABSOLUTE and PHYSICALRECORDS,
COLUMNPERCENT, ROWPERCENT, PHYSCOLPERCENT, PHYSROWPERCENT,
MEAN, PHYSMEAN,
MEANCUT,
MEDIAN, PCNTL1 - PCNTL4,
SUM, STDDEV, VARIANCE.

An example:

```
#macro #set( &1 )
cellset set1 :usevartitle &1 ( &1 &1_top &1_bot )
#endmacro

cellset set1( x x_top x_bot ) = absolute( x ) mean( x_top ) mean(
x_bot ) physicalrecords( x ) physmean( x );
descriptionstring =
"Die Zellen enthalten: gew. Basis, |Top-2-Box, Bottom-2-Box, |ungew.
Basis, |ungew. Mittelwert";

table = #k  by
#set ( v1 "Spiegel" )
#set ( v2 "Die Zeit" )
#set ( v3 "FAZ " )
#set ( v4 "die tageszeitung " )
#set ( v5 "Stern" )
#set ( v6 "Tagesspiegel " )
#set ( v7 "Süddeutsche" )
#set ( v8 "Handelsblatt " )
;
```

Übersichtstabelle
mit neuem CELLELEMENT CELLSET

Die Zellen enthalten: gew. Basis, Top-2-Box, Bottom-2-Box, ungew. Basis, ungew. Mittelwert	Total	Geschlecht		Irgendein unsinniger Kopf			
		männlich	weiblich	alle	fast alle	ein paar	wenige
N	346	179	168	346	321	227	53
Spiegel	317	164	153	317	295	215	52
	38,4	38,5	38,3	38,4	39,2	23,5	23,1
	47,8	45,3	50,4	47,8	46,4	63,0	41,0
	239	124	115	239	222	162	39
	2,4937	2,0806	2,9391	2,4937	2,4730	2,7778	3,2821
Die Zeit	254	134	120	254	237	193	45
	24,9	23,4	26,5	24,9	25,2	17,1	8,1
	43,6	50,2	36,8	43,6	44,0	62,8	5,4
	191	101	90	191	178	145	34
	3,1204	3,1485	3,0889	3,1204	3,1124	3,2759	4,5882
FAZ	321	170	152	321	297	210	48
	39,3	39,2	39,5	39,3	38,6	39,9	33,3
	47,0	46,7	47,4	47,0	48,0	44,3	58,3
	242	128	114	242	223	158	36
	3,0083	3,0156	3,0000	3,0083	3,0179	3,0127	3,0833
die tageszeitung	320	167	153	320	295	209	51
	40,5	41,8	39,1	40,5	38,7	41,4	50,0
	47,4	47,0	47,8	47,4	49,1	45,9	39,5
	241	126	115	241	222	157	38
	2,9544	2,9127	3,0000	2,9544	2,9865	2,9873	2,6579
Stern	312	170	142	312	290	207	49
	40,1	39,2	41,1	40,1	39,4	40,4	40,5
	47,5	47,5	47,7	47,5	48,2	46,8	51,4
	235	128	107	235	218	156	37
	2,9277	3,0234	2,8131	2,9277	2,9220	2,9487	2,8649
Tagesspiegel	312	162	150	312	290	205	48
	37,1	32,9	41,6	37,1	38,1	34,4	41,7
	51,4	53,1	49,6	51,4	50,0	55,2	47,2
	235	122	113	235	218	154	36
	3,0426	3,1967	2,8761	3,0426	3,0413	3,0714	3,0278
Süddeutsche	325	168	157	325	299	215	47
	37,4	34,4	40,7	37,4	37,3	37,0	42,9
	46,2	49,8	42,4	46,2	46,7	45,7	40,0
	245	127	118	245	225	162	35
	3,0327	3,0630	3,0000	3,0327	3,0356	3,0741	2,9143
Handelsblatt	311	165	146	311	293	205	51
	37,2	40,3	33,6	37,2	36,8	38,3	34,2
	45,3	45,2	45,5	45,3	45,0	41,6	44,7
	234	124	110	234	220	154	38
	3,0641	2,9516	3,1909	3,0641	3,0773	3,0649	3,2105

GESS mbH — Test CELLSET 2009/12/18 13:17:20

Analogue to the creative additions of the statistic row, e.g. `MEAN :FORMAT '#,##' ( x )` `FOREGROUND` and `USEFONT` can be used to expand the definition of `CELLSET FORMAT`.
e.g.

```
CELLSET MySet( p1 p2 p3 p4 ) =
mean :usefont "Helvetica" size 7 ( p1 )
mean :usefont "Helvetica" size 12 ( p2 )
mean :format '#,###' ( p3 )
sum ( p4 )
;
```

This information is then analysed when printing the individual data rows. `:DESCRIPTION` and `:USEVARTITLE` are not analysed.

FRAMEELEMENTS

TABLETYPEs are allocated to specific frame elements of a table; thus e.g. a table with row percentages (TABLETYPE = ROWPERCENT;) has by default an absolute column ("No. of Cases") and a total row ("Total"). With the specification FRAMEELEMENTS frame elements can be specifically requested. The key words necessary are:

```
ABSCOLUMN
ABSROW
PHYSICALCOLUMN
PHYSICALROW
TOTALCOLUMN
TOTALROW
```

ABSROW and ABSCOLUMN stand for rows (ROW) or columns (COLUMN) with absolute values of the cases or punches where relevant after weighting. PHYSICALROW or PHYSICALCOLUMN refer to the physical case number, i.e. without weighting. In TOTALROW or TOTALCOLUMN all the values for all the cases evaluated are printed as they have been defined in CELLELEMENTS.

Example:

```
FRAMEELEMENTS = ABSCOLUMN ABSROW TOTALCOLUMN;
```

With

```
FRAMEELEMENTS =;
CELLELEMENTS = ABSOLUTE;
```

e.g. a table is generated which not only shows the absolute frequency in the cells, but also contains no marginal distributions at all. FRAMEELEMENTS sets (as does TABLETYPE) a presetting for all the following tables which is valid until a new FRAMEELEMENTS- or TABLETYPE command follows.

If a current table frame is only to be altered the +/- Syntax analogue to TABLEFORMAT can be used:

e.g.:

```
FRAMEELEMENTS = -ABSROW;
```

only disables the row with the absolute values; everything else remains the same.

Usually TOTALROW or TOTALCOLUMN cells contain exactly those elements which the global CELLELEMENTS statement or the CELLELEMENTS option for the table has named in the TABLE statements. It could be desired to have for example additional contents in the TOTALROW. This is made possible by an explicit designation of the CELLELEMENTS for TOTALROW or TOTALCOLUMN.

Syntax:

```
CELLELEMENTS [ TOTALROW  | TOTALCOLUMN  ] = { <cellelement> }*n ;
```

e.g.:

```
CELLELEMENTS = COLUMNPERCENT;
CELLELEMENTS TOTALROW = PROJECTION COLUMNPERCENT;
```

This would cause the total row of a table which generally only contains the COLUMNPERCENT to additionally contain the PROJECTION.

III The CELLELEMENTS for Significance Tests

There is a series of CELLELEMENTS for significance tests whose results are presented in the form of letters which point to the columns or rows to be compared. Usually A refers to the first column, B to the second etc. The CELLELEMENTS are as follows:

Tests for Percentage Differences:

COLCHIQU Test per column for significant differences using 4-field-chi-square	ROWCHIQU Test per row for significant differences using 4-field-chi-square
COLPERCZ Test per column for significant differences according to arc-sine transformation using standardised normal distribution [2]	ROWPERCZ Test per row for significant differences according to arc-sine transformation using standardised normal distribution (see footnote 2)
COLPERCT T-Test per column for percentages on the basis of column overlaps	ROWPERCT T-Test per row for percentages on the basis of column overlaps
COLPERCEQUAL Tests all column percentages in one column for equality. Test according to arc-sine transformation using standardised normal distribution (see footnote 2)	ROWPERCEQUAL Tests all row percentages in one row for equality. Test according to arc-sine transformation using standardised normal distribution (see footnote 2)
COLPERCANDSIGN Column percent and COLPERCT	

Tests for Mean Differences:

TTEST Independent t-test (per column)	
MEANTEST Printing of mean value and t-test per column in one cell.	ROWMEANTEST Printing of mean value and t-test per row in one cell.
COLDEPTTEST Dependant t-test (per column)	
MEANCOLDEPT Printing of mean and the dependant t-test in one cell (per column)	

For all these CELLELEMENTS described in the above table the following options are available: SIGNIFLEVEL, SIGNIFTEXT, SHOWSIGNIF, TESTCOLUMNS and INDEXCHARS. Furthermore using a special variant of the COLOR statements a cell which has been appointed a letter due to significance can also be colour-coded.

SIGNIFLEVEL

Syntax:

```
SIGNIFLEVEL = <option>;
```

The key words and the relevant combinations of the levels of significance to be tested can be found in the following table:

[2] Minimum basis for this test is n = 25.

	0,1%	1%	5%	10%	20%
STDSIGNIFICANCE		X	X		
SIGNIF90				X	
SIGNIF95			X		
SIGNIF99		X			
SIGNIF999	X				
LOWSIGNIFICANCE			X	X	
HIGHSIGNIFICANCE	X	X			
ALLSIGNIFICANCE	X	X	X	X	
SIGNIF3LEVELS		X	X	X	
SIGNIF20AND5			X		X
SIGNIF20AND10				X	X

The key words are (apart from STDSIGNIFICANCE) identical to the TABLEFORMATS which have been used up until here. Here is a further one:

```
SIGNIFLEVEL = UNDEFINED; (preset)
```

The following rules apply to depiction:

Command	Depiction
STDSIGNIFICANCE	ABC... for 1%-Level, abc.... for 5%-Level
SIGNIF90	ABC... for 10%-Level
SIGNIF95	ABC... for 5%-Level
SIGNIF99	ABC... for 1%-Level
SIGNIF999	ABC... for 0.1%-Level
LOWSIGNIFICANCE	ABC... for 5%-Level, abc.... for 10%-Level
HIGHSIGNIFICANCE	ABC... for 0.1%-Level, abc.... for 1%-Level
ALLSIGNIFICANCE	A*B*C*.... for 0.1%-Level, ABC.... for 1%-Level a*b*c*.... for 5%-Level, abc.... for 10%-Level
SIGNIF3LEVELS	ABC... for 1%-Level, abc.... for 5%-Level, (a) (b) (c).... for 10% Level
SIGNIF20AND5	ABC... for 5%-Level, abc.... for 20%-Level
SIGNIF20AND10	ABC... for 10%-Level, abc.... for 20%-Level

AUTOSIGNIFTEXT

Syntax:

```
AUTOSIGNIFTEXT = [ YES | NO ];
```

If this is set to YES then whenever SIGNIFLEVEL is set and a significance test per column is available the relevant significance is printed in the BOTTOMTEXT. If there is no BOTTOMTEXT then one is created. If this is set to NO then in harmony with older versions the significance test is only printed if there is a BOTTOMTEXT.

SIGNIFTEXT

Syntax:

```
SIGNIFTEXT <option> = „Text zur Kennzeichnung“;
```

e.g.:

```
SIGNIFTEXT SIGNIF90 = „ABC ... for Signifikanz auf dem 10%-Niveau“;
```

There is a standard text for all options shown above. The SIGNIFTEXT command serves to modify this standard text. ***GESS* tabs** uses this text depending on the set SIGNIFLEVEL for the table and adds it to the end of an existing BOTTOMTEXT. This occurs only when the table contains a BOTTOMTEXT and if the SIGNIFLEVEL is unequally UNDEFINED.

The advantage of this control should be obvious: as the SIGNIFLEVEL is unambiguous the software can automatically allocate the relevant text and an incorrect allocation should be impossible.

The CELLELEMENTS mentioned above are tested against all cells in a row in a test per column. The identifying letters can in this (most usual) case be set automatically if TABLEFORMAT AUTOSIGNCHAR is set.

Alternatively all cells in a column are tested against the other cells in the same column if it is a test per row. Then the INDEXCHAR refer to the individual rows. In this case the identifying INDEXCHARS have to be set in the labels by hand; AUTOSIGNCHAR does not come into play here.

SHOWSIGNIF

Be it that a test resulted in a significant difference between column A and column D, then naturally the test between column D and column A would also show a significant difference. The identification of "A" in column D and of "D" in column A is technically correct but nonetheless redundant. In many cases it is preferable to show the significance only once for each pair. Up until now this was defined using TABLEFORMATS.

The new definition by means of the statement SHOWSIGNIF has taken its place. In the past the different conventions have arisen for testing means and percentages. These can also be defined using the new syntactical elements.

For testing mean differences:

```
SHOWSIGNIF MEAN = [ BOTH | LESS | GREATER | ANDEFINED ];
```

For testing percentage differences:

```
SHOWSIGNIF PERCENT =
[ BOTH | GREATER | LESS | ABSOLUTE GREATER | ANDEFINED ];
```

Analogue to SIGNIFLEVEL, UNDEFINED serves to handle TABLEFORMATS using the old control logic.

Colour-coding of significant coherence:

***GESS* tabs** provides the possibility of colour-coding the table FOREGROUND or BACKGROUND according to value ranges. As it is often wise to identify particularly those cells where significant deviances have been recorded the successful significance test is allocated a "synthetic" value of 99999.

Using the COLOR statement a reaction to the result of a specific significance test is possible, e.g.:

```
RGB = NO;
COLOR FOREGROUND =
| COLCHIQU RANGE 99998.9  99999.1 : 0 1 1
;
```

This causes all significance letters to be red COLCHIQU.

COLPERCTMINIMUM

In the significance calculation using COLPERCT the column overlaps are taken into account. This method can lead to problematical significances if the number of overlaps is so high that there are only a few cases which do NOT occur in both columns which have been tested against each other. If COLPERCTMINIMUM is set to zero the test result in the column pairs in which at least that number of cases (sum of the weights) do not overlap is not printed

TESTCOLUMNS

Syntax:

```
TESTCOLUMNS = { Testdefinition }*n;
Testdefinition ::= | VARIABLE <varno> CODE <code> : VARIABLE <varno>
CODE <code>
```

If no TESTCOLUMNS are set then all the columns are tested against each other per variable. If the test definition remains empty then all the columns are tested against each other independent of the variable.
e.g.:

```
TESTCOLUMNS =
| VARIABLE 1 CODE 1 : VARIABLE 2 CODE 1
| VARIABLE 1 CODE 2 : VARIABLE 2 CODE 2
;
```

In this example only two tests are carried out; the Code 1 of the first variable in the stub against Code 1 of the second variable in the stub; and the same for Code 2 in both variables.

or, in order to test all the columns independent of the variables:

```
TESTCOLUMNS =  ;
```

INDEXCHARS
e.g.

```
INDEXCHARS = "GEHT";
```

allocates a (small or large) G to the first test column, an E to the second, an H to the third and a T to the fourth. The letters A – Z are preset.

TESTCOLUMNS are taken into account.

The letters A – Z can initially be used as INDEXCHARS to deal with 26 columns. If more are needed, this is also possible, however for all other symbols, lower and upper case allocations are unknown and have to be set explicitly. The command for this is LOWERCASE. (see below)

LOWERCASE
Syntax:

```
LOWERCASE <char> = <char>;
<char> ::= [ x | 'x'  | "x" | <number> ]
x ::= A .. Z, a .. z
number ::= 1 .. 255
```

Normally only the letters A – Z can be used in INDEXCHARS, as there are only signs (ASCII Code < 128) for these codes for stipulating upper or lowercase letters in standard software. As ***GESS* tabs** also allows the free allocation of printable symbols to codes there is no automation possible here. Therefore the allocation of lower or upper case letters can be defined in the script.

For a typical German case the following is used e.g.:

```
LOWERCASE Ä = ä;
```

Therefore there can also be significance tests in tables with more than 26 columns. :-(

The test results of tests per column (COLPERCZ, COLPERCT, COLCHIQU) are usually undertaken between all columns of **one** variable; here the columns are identified one after the other with a letter that has been defined in INDEXCHARS. Columns in other variables can thus not be tested. Alternatively the columns to be tested can be explicitly defined using TESTCOLUMNS causing only the explicitly named test to be carried out; this can then occur regardless of the variable.

AUTOSIGNFORMAT

Syntax:

```
AUTOSIGNFORMAT = "<formatstring>";
```

In the usual case only the significant letter is used in the label box. If this is to be made more attractive a special format string is required. All the symbols in the string are used; # in the string is replaced by the INDEXCHAR.

BENCHMARKVALUES

A different subject: percentage values and bases (e.g. from other surveys) can be set in BENCHMARKVALUES. Then the column percentages in the relevant cells are compared with the benchmark values using a z-test. The BACKGROUND of the cell can then be coded with BENCHMARKCOLOR.

A new product, for example, that has been subjected to a standard test can also be compared with the mean of a similar product from an earlier test.

BENCHMARKVALUES must always come AFTER the TABLE statement. In the TABLE statement CELLELEMENTS COLPCTBENCHMARK must take the place of COLUMNPERCENT.

If the percentage values which form the basis of the benchmark test are also to be printed in the table (as in the example below) then the HISTORY statement is required.

Syntax:

```
BENCHMARKVALUES
| VarNoX ValuesX / VarNoY ValuesY = <Prozentwert> <Basis>
....
....
;

e.g.:
BENCHMARKVALUES
| 1 1:3 / 1 1:1 =  8.45  9213
| 1 1:3 / 1 2:2 = 54.33  9213
;
```

The first line of the example means: the benchmark percentage value 8.45 for an N of 9213 is valid for all cells in the point of interception of the first variable in the X-axis with the x values 1 2 and 3 and the first variable in the Y-axis with the value 1.

BENCHMARKCOLOR

Syntax:

```
BENCHMARKCOLOR = <color_high> <color_low>  ;
```

Two colour values are required (RGB or HSB), for the percentage values above the benchmark value and for those below.
e.g.

```
#expand #high 1 0.0.9 0.9
#expand #low  0.9 0.9 1.0
RGB = YES;
BENCHMARKCOLOR = #high #low
```

BENCHMARKLEVEL

Syntax:

```
BENCHMARKLEVEL = [ SIGNIF90 | SIGNIF95 | SIGNIF99 | SIGNIF999 ];
```

Here is an example of how this could look:

Neuprodukte im Vergleich zur bestehenden Produktpalette

Neue Shampoos im Benchmarktest in sechs Testdimensionen.

	Shampoo X	Shampoo XX	Shampoo R1	Shampoo T6	Shampoo F	Shampoo A	Bench-mark
Basis 100%	**789**	**787**	**787**	**787**	**787**	**785**	**1675**
Top Two Box Glanz	59,1 %	59,0 %	57,5 %	60,3 %	60,0 %	50,2 %	57,6 %
Top Box Glanz	33,6 %	33,9 %	33,0 %	35,4 %	33,5 %	27,1 %	27,1 %
Top Two Box Fülle	20,5 %	18,8 %	25,3 %	27,2 %	18,8 %	21,0 %	25,3 %
Top Box Fülle	1,5 %	1,1 %	1,7 %	1,7 %	1,7 %	1,1 %	3,9 %
Top Box Verpackung	38,6 %	27,4 %	32,0 %	35,9 %	36,3 %	32,7 %	33,9 %
Top Box Qualität	48,1 %	33,9 %	44,1 %	41,3 %	48,8 %	41,1 %	42,3 %

Farben:
Rot = signifikant unterhalb des Benchmarks / Grün = signifikant oberhalb des Benchmarks (5% Niveau)
Blau: Benchmark Werte

Software
by GESS

TABLEBASE

This controls the basis of percentaging in the `TABLE` printout. The following is preset:

```
TABLEBASE = CASES ;
```

i.e. usually percentaging is on the basis of the number of interviewees. Using

```
TABLEBASE = NOMINATIONS ;
```

the alternative of percentaging on the basis of the number of mentions can be achieved (only relevant for multiple responses). Using `CODEBOOK` tables always causes the number of interviewees to be the basis of percentaging. The `TABLEBASE` setting remains valid until a new `TABLEBASE` is defined.

USECASES

`USECASES` controls the treatment of `MISSING` values in cross tables. Usually the rows and columns of cross tables are suppressed if either no `VALUELABEL` has been defined or if the relevant characteristic in a `MISSING` command has been declared a `MISSING` value, or if a characteristic is recognised as a `MISSING` value due to explicit coding (see `MISSINGCHAR`).

A case will only be placed in the table body if a valid value is present for both variables that have been crossed. The calculation in the table frame (see `FRAMEELEMENTS`) and in the total number of cases can however be influenced by the setting of `USECASES`. `MISSING` is the only relevant characteristic for the evaluation of a case; the existence of a label is in this case of little interest. (For old hands at ***GESS* tabs**: this new rule is valid as of ***GESS* tabs** 2.10; in earlier versions unlabelled values were also invalid for "..`VALID`"-conditions.)

As a rule:

- a case is counted (appears in the frame and thus in the percentaging base) if it is deemed valid by the `MISSING` values according to the constellations below or if the table is counted using the stipulation `USEMISSING = YES`;

- a case will be printed (appears in the table body) if additionally a valid `VALUELABEL` for both dimensions is also present or `PRINTALL = YES` is set for the relevant variable.

```
USECASES = ANYCASE ;
```
All cases flow into the table total-N unless they have been excluded by SELECT or TABSELECT. All cases which are valid for the relevant variable flow into the frame cells. The individual columns and rows therefore will not necessarily add up to 100% or the total for the column or row.

```
USECASES = XANDYVALID ;
```
This setting is the absolute opposite of ANYCASE. Only if a case is valid in the X and the Y variable will it be counted ("X and Y Valid"). Only the cases which have valid values for the variables in the x-axis and the variables in the y-axis flow into the table total-N. X and Y axes are thus treated symmetrically. Only the cases which have a valid value for the x-axis flow into the frame cells of the Y-variable and vice versa.

```
USECASES = XORYVALID ;
```
("X or Y Valid"). Every case which either has a valid X value or a valid Y value flows into the total-N. All the valid values of the marginal distribution flow into the relevant marginal distribution. This is also symmetrical for X and Y.

```
USECASES = XVALID ;
```
("X Valid"). All the cases which have a valid value for the x variable flow into the total-n. All the cases which have a valid value in the x variable flow into the marginal distribution of X. All the cases which have a valid X value and also a valid Y value flow into the marginal distribution of the Y variable. This option treats the X and Y axes differently; it is an asymmetrical option.

```
USECASES = YVALID ;
```
("Y Valid"). The asymmetrical counterpart to XVALID. All the cases which have a valid value for the y variable flow into the total-n. All the cases which have a valid value in the y variable flow into the marginal distribution of Y. All the cases which have a valid X value and also a valid Y value flow into the marginal distribution of the X variable.

Comment on marginal distribution in nested tables:
The preset for USECASES generally functions as described above in nested tables but it is more complex. In for example the "TABLE = a b BY c d;" there are four marginal distributions which can differ i.e. for the logical tables a by c, b by c, a by d and b by d. The USECASES rules are valid for each logical table; i.e. the marginal distribution from a can be different in the table "a by c" to the one in the table "a by d" because there could be different cases valid in the variable c to those in variable d. The correct marginal distribution is calculated and used for percentaging within the "undertitles". For each of these tables there can be a different table total-N.

There can however only be one marginal distribution printed in the table frame. According to ***GESS* tabs** convention it is the marginal distribution which results from the crossing of each of the first variables from the other direction. In nested tables percentage bases can occur which are not visible in the marginal distribution. If this can cause misunderstanding it is advisable to explicitly include filtered absolute columns and rows (ABSOLUTE (Varname)), or to pull the tables apart and present them separately.

In the following example the use of USECASES = YVALID will be demonstrated to combine two filtered sub-tables with a filtered variable. The following steps are of note:

1. variables v6 and v7 are filtered dependent on v5: i.e. they are guaranteed MISSING where they are not to be asked.

2. USESCASES is set to YVALID. this ensures that the marginal distribution only contains those cases in the sub-tables where the variables in the Y-axis are filtered (v6 and v7 are in the Y-dimension of the table),

3. relevant marginal distribution is printed as ABSOLUTE underneath the table: once as all cases (filtered with the constant ALL) and for users (ben) and non-users (nben). ben and nben are formed from v5.

4. (typo-)graphical presentation is taken from `FMT05A.FMT` .

```
// 4.
INCLUDE = FMT05A.FMT;
SINGLEQ v5 = TITLE "Nutzung des ÖPNV" 1
LABELS
1 "ÖPNV#Nutzer"
2 "ÖPNV#Abstinenzler"
;
// Zur Filterung der Absolutverteilungen werden entsprechende
variables
// gebildet. Diese müssen immer dann MISSING sein, wenn kein Fall in
die
// Sub-Tabelle eingeht.
COMPUTE alle = 1;                  TITLE = "Basis (Alle Befragten)";

// nie missing
// 3.
COMPUTE ben = v5;  MISSING = 2;  TITLE = "Basis (ÖPNV#Nutzer)";

// Bei Nicht-Nutzern Missing
COMPUTE nben = v5; MISSING = 1;  TITLE = "Basis (ÖPNV#Abstinenzler)";
// Bei Nutzern Missing
SINGLEQ v6 = TITLE "an Nutzer:\wichtigster Grund for ÖPNV Nutzung"
2
LABELS
1 "Umweltschonung"
2 "Preis"
3 "Nervenschonung"
0 "K.A."
;
// 1 .
FILTER = v5 EQ 1;       // Nur bei Nutzern gültig
SINGLEQ v7 = TITLE "an Nicht#Nutzer:\wichtigster Grund zur Ablehnung
des ÖPNV"
3
LABELS
1 "zu langsam"
2 "zu teuer"
3 "zu unangenehm, zu dreckig, zu laut"
;
// 1.
FILTER = v5 EQ 2;       // Nur bei Nicht-Nutzern gültig
SINGLEQ Alter =
26
LABELS
1 "bis 24"
2 "25 bis 29"
3 "30 bis 39"
4 "40 bis 60"
5 "über 60"
;
// 2.
USECASES = YVALID;

TABLEFORMAT = NOBODYBLANKS;
TABLETITLE =
```

```
"Nutzer und Nicht#Nutzer des ÖPNV:\ein Beispiel für gefilterte
Subtabellen";
TABLETYPE = COLUMNPERCENT;
FRAMEELEMENTS = TOTALCOLUMN;

TABLE = alter BY v5 ABSOLUTE( alle ) v6 ABSOLUTE( ben ) v7 ABSOLUTE(
nben ) ;

TOPTEXT = "
Frage an Nutzer:
Was ist für Sie der wichtigste Grund, den ÖPNV zu nutzen?

Frage an Nicht Nutzer:
Was ist Ihr wichtigster Grund, den ÖPNV nicht zu nutzen?
";
```

Nutzer und Nicht-Nutzer des ÖPNV: ein Beispiel für gefilterte Subtabellen

Frage an Nutzer:
Was ist für Sie der wichtigste Grund, den ÖPNV zu nutzen?

Frage an Nicht-Nutzer:
Was ist Ihr wichtigster Grund, den ÖPNV nicht zu nutzen?

		Alter				
Sp. %	*Insge-samt*	bis 24	25 bis 29	30 bis 39	40 bis 60	über 60
Nutzung des ÖPNV						
ÖPNV-Nutzer	*51,1 %*	40,0 %	53,3 %	50,7 %	55,8 %	50,0 %
ÖPNV-Abstinenzler	*48,9 %*	60,0 %	46,7 %	49,3 %	44,2 %	50,0 %
Basis (Alle Befragten)	*650*	90	150	146	172	92
an Nutzer: wichtigster Grund für ÖPNV-Nutzung						
Umweltschonung	*25,3 %*	33,3 %	27,5 %	18,9 %	33,3 %	8,7 %
Preis	*50,6 %*	33,3 %	35,0 %	59,5 %	54,2 %	69,6 %
Nervenschonung	*23,5 %*	33,3 %	35,0 %	21,6 %	12,5 %	21,7 %
K.A.	*0,6 %*	0,0 %	2,5 %	0,0 %	0,0 %	0,0 %
Basis (ÖPNV-Nutzer)	*332*	36	80	74	96	46
an Nicht-Nutzer: wichtigster Grund zur Ablehnung des ÖPNV						
zu langsam	*33,3 %*	29,6 %	37,1 %	36,1 %	34,2 %	26,1 %
zu teuer	*43,4 %*	25,9 %	42,9 %	55,6 %	50,0 %	34,8 %
zu unangenehm, zu dreckig, zu laut	*23,3 %*	44,4 %	20,0 %	8,3 %	15,8 %	39,1 %
Basis (ÖPNV-Abstinenzler)	*318*	54	70	72	76	46

GESS mbH Beispiele

TABLESTATISTICS
alias
STATISTICS

The definition for presets for the required statistical values for all table parts where there are no explicit statistical commands. Should for example the chi-square and the contingent coefficiency be calculated as standard for all tables:

```
TABLESTATISTICS = ( CHIQU CONTINGENCY );
```

The required code is always placed in brackets. The program calculates the following codes:

CHIQU	Chi-square
CRAMERSV	Cramers V
TAUB	Tau B
TAUC	Tau C
CONTINGENCY	contingent coefficiency, standardised
CONTINCENCYNONSTD	contingent coefficiency, non-standardised
SOMERSDCOL	Somers D, regardless of column
SOMERSDROW	Somers D, regardless of row
SOMERSDSYM	Somers D, symmetrical
GAMMA	Gamma

USERAWSFORSTATS

Syntax:

```
USERAWSFORSTATS = [ YES | NO ] ;
```

Usually the statistical codes above (STATISTICS) are calculated on the basis of the weighted table cells: i.e. comprehensible for ABSOLUTE tables. These codes can also be calculated on the basis of the unweighted cells if so desired.

MARKCELLS

Syntax:

```
MARKCELLS = [ YES | NO ] [ COLOR {colors}*6 | CELLELEMENTS
<cellelement> ];
```

This sets the identification of cells which vary significantly from the marginal distribution. It only investigates absolute cells and not mean of variance differences. The test bases on a 4-field-chi-square. For this test the table is broken down into all possible 4-field tables. A 3 x 4 table e.g. can be broken down into 12 4-field tables. As of a certain probability of error <= 5% the field is shaded grey; the depth of the shade depends on the usual stages (5%, 1%, 0.1%) according to probability of error. Only those cells are shaded which have at least one of the following CELLELEMENTS: COLUMNPERCENT, ROWPERCENT or ABSOLUTE.

There is a choice of 6 colour codes (RGB or HSB) which can be placed before the semicolon after the key word COLOR and thus define the shading of the cells.

Alternatively the key word CELLELEMENTS can be used; instead of shading there are between 1 and 3 plus or minus signs (depending on significance) attached to the relevant cell element.

Regarding the colour codes:

It is usually easier to use HSB (Hue, Saturation, Brightness) to highlight different levels, i.e. the same hue is used three times for all cells which are greater or smaller and only the saturation is varied.

e.g.:

```
#expand #lt050 0.0 0.10 1.0
#expand #lt010 0.0 0.20 1.0
#expand #lt001 0.0 0.30 1.0
#expand #gr050 0.6 0.10 1.0
#expand #gr010 0.6 0.20 1.0
```

```
#expand #gr001 0.6 0.30 1.0

RGB = NO;
MARKCELLS = YES COLOR
#lt050          // %5        kleiner als der Durchschnitt
#lt010          // 1%        kleiner als der Durchschnitt
#lt001          // 0,1%      kleiner als der Durchschnitt
#gr050          // %5        grösser als der Durchschnitt
#gr010          // 1%        grösser als der Durchschnitt
#gr001          // 0,1%      grösser als der Durchschnitt
;
```

CELLMINIMUM

The option CELLMINIMUM states as of which minimum value a table cell counts as valid and should be included.

Example:

```
CELLMINIMUM = 10;
```

In all cells where the minimum value has not been reached there will be "-". Preset at 0.0001;

CELLMINIMUM as ROWMINIMUM and COLMINIMUM are TABLE options. Options always refer to the last table requested. They are therefore always written after the TABLE command. If the option is written before the first TABLE command this leads to a syntactical error. The options serve to suppress "uninteresting" cells, rows or columns in the table. Empirically vacant rows or columns can be printed by setting ROWMINIMIM or COLMINIMUM to zero.

ROWMINIMUM

Option for TABLE statement. Only those rows are printed which contain at least ROWMINIMUM cases, i.e., characteristics with very low case numbers in side group variables are suppressed. Preset at 0.0001.

GLOBALROWMINIMUM

Global preset for ROWMINIMUM for all following tables.

COLMINIMUM

Option for TABLE statement. Only those columns are printed which contain at least COLMINIMUM cases, i.e., columns with very low case numbers in side group variables are suppressed. Preset at 0.0001.

ROWCELLMINIMUM

Syntax:

```
ROWCELLMINIMUM = <number>;
```

Post nominal option for the last TABLE or XTAB statement. A table row is suppressed if not at least one cell has an absolute value of <number>.

COLPERCENTLINELIMIT

Syntax:

```
COLPERCENTLINELIMIT = <number>;
```

Parallel to the option above, a row is suppressed if a cell has a column percent value of <number>.

MINCOLBASE

Syntax:

```
MINCOLBASE = <number>;
```

Preset at `MINCOLBASE = 0`

`MINCOLBASE` is used to set a minimum column value below which the column percent is not printed. If a column has fewer cases (calculated as the sum of the weights) than stipulated by `MINCOLBASE` the column is still printed but the column percentages are suppressed (or replaced by a blank).

GLOBALCOLMINIMUM

Global preset for `COLMINIMUM` for all the following tables.

TABLEFORMATs for TABLE Tables

TABLEFORMAT
The table appearance can further be controlled using TABLEFORMAT.
Syntax:

```
TABLEFORMAT = [ + | - | ] { Formatoption ... }*n ;
```

If the assigning symbol is followed by plus or minus the current TABLEFORMAT setting is kept and the following format options are added (+) or removed (-) from the valid format. If a format option directly follows the assigning symbol the current TABLEFORMAT setting is overwritten by a list of the named format options.

The following TABLEFORMATs are valid for TABLE tables (and below for XTAB tables):

AUTOSIGNCHAR
This TABLEFORMAT ensures an automatic identification of the column with an identifying letter (see INDEXCHARS) for significance tests per column. If TESTCOLUMNS has been set the letters are not re-allocated for each variable as is usually the case.

AUTOSIGNCHARALWAYS
As AUTOSIGNCHAR but using AUTOSIGNCHAR the identification in the stub automatically only occurs if also at least one valid CELLELEMENT is present in the table. This test does not take place if using AUTOSIGNCHARALWAYS.

TEXTWRAP
Usually the variable texts are presented exactly as they have been defined. TEXTWRAP is used to break up the lines in text boxes.

MULTITOTALX
Usually the TOTALROW is counted on the basis of case numbers (see also TABLEBASE). In many cases it is required to have a total different to the number of response for variables with multi-responses. This can be done using TABLEFORMAT. (e.g. 165% in the total row of a column percentage means an average of 1,65 responses per interviewee.)

In this context:
For VARGROUPs and VARFAMILYs the multi-responses are required in the total on the one hand; individual characteristics (hierarchically placed above or below) should be omitted on the other. This is achieved by allocating LEVEL <> 0 to the characteristics.

PAGETOTALX
Only effective with MULTITOTALX: The responses of all variables on the Y-axis are tallied for the TOTALROW.

MULTITOTALY
Analogue to this a TOTALCOLUMN is usually tallied on the basis of number of cases. Using MULTITOTALY this tally can be converted to all responses.

PAGETOTALY
Only has effect with MULTITOTALY: The responses to all variables on the X-Axis are tallied for the TOTALCOLUMN.

ABSINLABELBOX
Prints the ABSOLUTEROW at the lower frame of the label box instead of as it is usually in its own box.

PERCENTINLABEL
Automatically adds a % symbol with CELLELEMENT = COLUMNPERCENT to the label boxes on the X-axis. Incidentally is also used to add an additional row with percentaging in ColumnCount (PS).

NOBODYBLANKS
Suppresses blank rows in the table body that have been added to improve legibility. Tables then may for example fit on one page.

NOHEADERBLANKS
Suppresses blank rows in the stub. (**NON-PS**)

NOVARTITLEBOX
Suppresses the box which names the variables on the Y-axis. Always makes sense if only one variable is used on the Y-axis which for example already appears in the TOPTEXT box.

NOCONTENTBOX
Suppresses the explanation box in additional table rows which for example contain mean or sum etc. In this case only the VARTITLE or the VARNAME are printed in front of the value. The user should then include other texts to explain the content. (no effect on Postscript-printouts). (**NON-PS**)

NODESCRIPTION
Suppresses the descriptive text for the cell contents (see DESCRIPTION.

MODIFYVARNAME
Prints not only the variable name but also the DESCRIPTION of the column or row content in columns or rows with third variables (e.g. MEAN (Einkommen)).

MEANDESCRIPTION
Replaces the variable name with a description string e.g. "mean" in columns and rows with third variables.

SUPPRESSLABEL
If a variable is a constant (i.e. it has empirically only one characteristic), it can make sense for appearances sake to suppress the label text. This can be achieved with SUPPRESSLABEL. (Only effective with Postscript-printouts). (**PS**)

EXPANDBOX
If a shared block has been drawn around the data cells using DRAWBOX it often looks better if there is a vertical space before the first and after the last data row and the upper and lower frames. This space can be set using EXPANDHEIGHT; it should be noted that then the DATABOX is not congruent to the sum of the DATACELLs. (Only effective with Postscript-printouts). (**PS**)

PRINT2LINES
The presentation of two rows within a cell can be achieved in rows or columns that are generated using CELLELEMENTS and have two logical contents (e.g. ABSCOLPERCENT, ABSMEAN). (Only effective with Postscript-printouts). (**PS**)

PRINT2LINES2
Analogue to Print2Lines, only in the other order. (Only effective with Postscript-printouts). (**PS**)

ADDOVERCODE
Usually the OVERCODE is only tallied once per case if several of the relevant categories arise i.e. a logical OR is used. ADDOVERCODE requests the addition of the individual frequencies.

LONGVARTITLE
Ensures that the VARTITLE in the table Y-Axis is not broken up. Should only be used if no DRAWBOX for VARTITLE Y has been defined. Otherwise it looks stupid! (**Only PS**)

SIMPLEPERCENTILE
TABLEFORMAT fully developed. See below.

PERCENTILEINTERPOL
Using this TABLEFORMAT an interpolation is switched on. Interpolation used to be standard in ***GESS* tabs**. This is however unusual if anything; we have readjusted and now interpolation has to be explicitly defined.

AUTOOVERSORT
Sorts the OVERCODES in a table and prepares the labels for sorting within the overcode. Overcodes can hierarchically be sorted on up to five levels.

NOZERODASH
Usually the real zero in percentage tables is represented by a "-". This can be switched off using NOZERODASH.

ABSZERODASH
Usually zero as an absolute value is represented with a "0". ABSZERODASH can be used to represent the zero in a CELLELEMENT ABSOLUTE as a dash ('-').

MULTICOLINHG
Multiple cell contents (e.g. ABSCOLPERCENT) in CSV-Data files are usually represented in several rows. Alternatively they can be presented in several columns using +MULTICOLINHG.

USEFORMATINHG	Formats for CELLELEMENTS are also adopted for printouts in HG.
TOTALCOLINHG	Total column adopted in HG
TOTALROWINHG	Total row adopted in HG
ABSCOLINHG	Absolute column adopted in HG
ABSROWINHG	Absolute row adopted in HG
PHYSCOLINHG	Absolute column unweighted adopted in HG
PHYSROWINHG	Absolute row unweighted adopted in HG
TABLETITLEINHG	Table title adopted in HG
TEXTBOXINHG	Toptextbox adopted in HG
VARNAMEXINHG	Variable title of the X-Axis adopted in HG
VARNAMEYINHG	Variable title of the Y-Axis adopted in HG

LOCALCONTENT
During printing the information is taken from the locally set cell contents and not from FRAME.

SLICELASTPAGE
Tables generated on the Y-Axis SLICE or LINESLICE (e.g. TABLE = y by b SORT POSITION SLICE 10 MEAN(b);) usually have the mean (or other value) on each page. This TABLEFORMAT ensures the printout only on the last page.

GLOBALSORT
Normally a SORT key word in a TABLE statement effects only the directly preceding dimension of a table:

```
TABLE = #kopf by a b sort absolute descend
```

e.g. only sorts within the characteristics of the variable b. GLOBALSORT enlarges the scope of SORT to the whole table. This particularly makes sense for mean tables etc.
Using:

```
TABLEFORMAT = + GLOBALSORT;
TABLE = #kopf by MEAN( a ) SORT MEAN
MEAN( b ) SORT MEAN
MEAN( c ) SORT MEAN
......
......
;
```

the whole table is sorted by mean. For technical reasons SORT MEAN has to be repeated after each MEAN(…).

HISTORY
Syntax:

```
HISTORY =
[ DATABOX <x> <y> ] FORMAT ( <Formatliste> ) DATA [ Absliste ] {
<number> : <Dataliste> }*n;

Formatliste                          ::= [ ABSROW | ABSCOLUMN |
PHYSROW PHYSCOLUMN TOTALROW ] { <number> }*n
Absliste  ::= { <number> }*n
Dataliste ::= [ <number> | ] { <string> }*n
```

HISTORY is an option for TABLE statements and refers to the last table. Using the HISTORY statement already known data that has not been generated in the current data file can be added to the data tallied in a particular table. These are often global results from earlier surveys (historical waves) that are to be tallied anew.

Table cells (as single strings) and either an absolute column or an absolute row can be defined. The key words are ABSCOLUMN or ABSROW in the format list. The format list describes the structure of the data listed behind the key word DATA.

If ABSCOLUMN is the first element in the format list every data row expects the first element after the colon to be the absolute basis of each row. If ABSROW is the first element then the absolute basis of each column is expected in a row before the first data row. All further format list elements are interpreted as numerical values for the columns.

Every data row is introduced by a numerical value for each data row followed by a colon and followed by the cell content for each further format list element. The cell content is adopted as a string; thus there can be for example a % sign. The cell contents may have to be put in inverted commas in order to mark them clearly.

Just as data cells in tallies of current tables are defined by pairs of variates, namely by the relevant variable values in the X- or Y-Axis, the data cells in the HISTORY command are also defined by pairs of variates. The value of the Y-variable is before the colon at the beginning of each data row, the value of the X-variable results from the format command as is shown clearly in the example below. The HISTORY statement is as all ***GESS* tabs** statements format free.

Using DATABOX individual table parts within a nested table can be referred to. If there is a table for example where the pre column contains first a frequency distribution and then a row with means like so:

```
TABLE = KOPF by Frage1 MEAN( Frage1.num );
```

then DATABOX 1 2 can refer to the row with the mean:

```
HISTORY = DATABOX 1 2 FORMAT ( 99 100 ) DATA
1 : 22.3  33.3 ;
```

The numerical arguments of DATABOX refer to the additional table elements in the X- and Y-axis. In the example above there is only one element in the x-axis, the variable "Kopf" whose characteristics 99 and 100 are seen as "historical". The means are the second element in the description of the pre column (after BY). That is why they are addressed with y=2.

An example in context:

```
COMPUTE Erhebungswelle = 5;
{ in the current data file the wave has a constant with the value of
5. The results for waves 1 - 4 are included using the HISTORY
statement }
VALUELABELS Erhebungswelle =
1 "Welle 1"
2 "Welle 2"
3 "Welle 3"
4 "Welle 4"
5 "aktuelle Erhebung"
;
Variable Noten = 27  { variable used is in column 27 }
LABELS
1 "sehr gut"
2 "gut"
3 "befrie-digend"
4 "aus-rei-chend"
5 "mangel-haft"
6 "unge-nügend"
9 "weiß nicht"
0 "K.A."
;
TABLE = Noten BY Erhebungswelle;
HISTORY =
FORMAT ( ABSCOLUMN
           1    2   3   4   5   6   9  0
) DATA
1 :  6424  5   34  27  12   5   1  10  6
2 : 14079  4   32  30  13   5   1  10  5
3 :  9897  5   32  30  14   4   1   9  5
4 :  9815  5   32  31  13   3   1  10  5
;
```

SORT

Normally the variable characteristics are printed in the order they are defined in VALUELABELS statement.

The variable characteristics in the X or Y-Axis can however also be sorted according to other criteria. The key word SORT is written after the variable name followed by the sort criterion which are as follows:

ABSOLUTE	acc. to absolute cell content
MEAN	acc. to arithmetical mean
SUM	acc. to cell sum
VALUE	acc. to numerical value (Code)
ALPHA	acc. to label text

If ALPHA is chosen as default this can lead to cases where no VALUELABELS have been defined. In this case ***GESS* tabs** generates internal labels which are presented in the table in ascending numerical order; thus in this case the default order is the same as sorting according to VALUE.

If only a few of the empirical values occurring are unlabelled then they appear at the end of the table.

ABSOLUTE, VALUE or ALPHA can always be used to sort SUM and MEAN can only be used if the state of the data allows i.e. the cell must contain a sum.

Usually SORT sorts according to the marginal distribution (or total distribution) of the first element in the table head e.g.:

```
TABLE = a BY b SORT ABSOLUTE DESCEND;
```

Using the key word PANE a different column can be taken as the sort basis, in the following case for example the mean of b in the second element of the stub:

```
TABLE = a MEAN( b ) BY c SORT MEAN PANE 2 CODE 1;
```

Using the key word CODE the sorting according to a single column with differing codes can be specified. The statement CODE is obligatory after PANE. In individual columns/rows with MEAN etc. the information is always found in the column under the (synthetic) Code 1; that is why "CODE 1" is in the example above.

Using PROZENTCOLUMN or ABSOLUTCOLUMN the data can be sorted according to the distribution under certain codes, e.g. the column with the Code 2 of the head variable a:

```
TABLE = a MEAN( b ) BY c SORT ABSOLUTE PANE 1 CODE 2;
```

This (more flexible) solution replaces the older ***GESS* tabs** sorting according to the invisible implied total distribution.

In connection with SORT the output can be confined to parts of the distribution; either the upper part (TOP), the lower part (BOTTOM) or both ends of the distribution (EXTREME) can be selected.
Example:

```
TABLE = a MEAN( b ) BY c SORT MEAN PANE 2 EXTREME 20; // 20 each
// from each end of the distribution
TABLE = a BY c SORT ABSOLUTE TOP 80;   // the top 80
```

There is a TABLEFORMAT specifically for SORTGLOBALSORT. Normally a SORT key word is only valid in the TABLE statement within the TABLEPART. Using GLOBALSORT its effect for summary tables made up of statistical coefficients e.g. mean is widened to the whole table.
Using:

```
TABLEFORMAT = + GLOBALSORT;
TABLE = #kopf by
MEAN( a ) SORT MEAN
MEAN( b ) SORT MEAN
MEAN( c ) SORT MEAN
......
......
;
```

the whole table is sorted according to the mean. For technical reasons the SORT command has to be written after every MEAN(…) etc. The key words DESCEND, PANE and CODE for specifying the sort are permitted.

Additionally the key word RANGE can be used to generate whatever areas are necessary to break down a table with many characteristics:

```
TABLE = a BY b SORT ABSOLUTE DESCEND RANGE  1 20;
```

If descending order is required then the key word `DESCEND` is written after the sort criterion.

Similar to `RANGE`, `SLICE` is used to divide a table along the y-axis into the number of individual tables required.

```
TABLE = a BY b SORT ABSOLUTE DESCEND SLICE 15;
```

This ensures that a table with e.g. 55 single items in `variable b` is divided onto four pages. If the split were to leave a page with just one response, this response is added to the previous page. A table with 61 items is thus printed on four and not on five pages.

```
TABLE = a BY b SORT ABSOLUTE DESCEND SLICE 15 TOP 30;
```

An additional (post-nominal) `TOP` statement allows a limitation to the first n (in the example 30) characteristics to be printed.

Using `LSLICE` (short for LineSLICE) a long table can be divided according to label row. This ensures a better balanced printout of tables with greatly varying label lengths.
Example:

```
TABLE = a BY b SORT ABSOLUTE DESCEND LSLICE 30;
```

This is then particularly important when long code plans for complexly coded open questions are to be tabulated. In this context we draw particular attention to `TABLEFORMAT AUTOOVERSORT`. If this code plan is to be sorted on tabulation then it will appear with the hierarchical structure of the code plan like so:

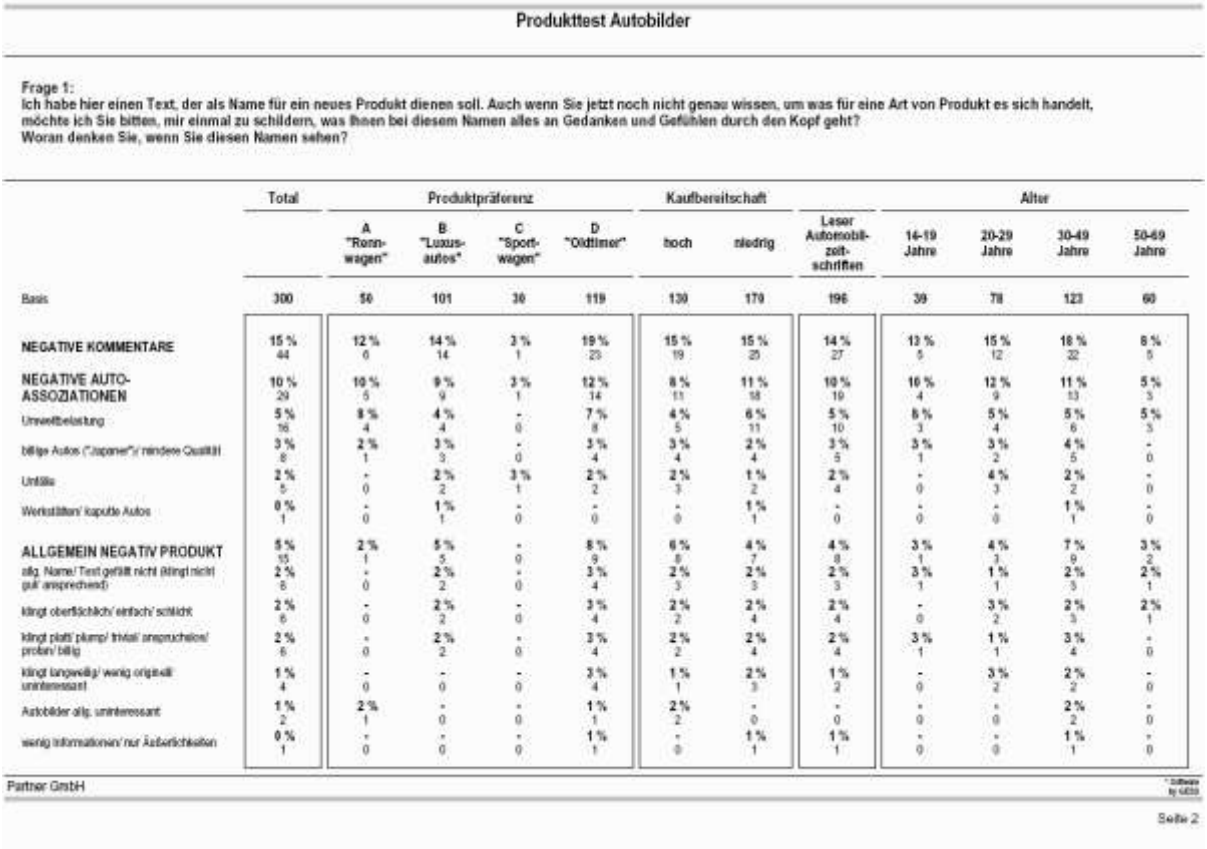
Produkttest Autobilder

Frage 1:
Ich habe hier einen Text, der als Name für ein neues Produkt dienen soll. Auch wenn Sie jetzt noch nicht genau wissen, um was für eine Art von Produkt es sich handelt, möchte ich Sie bitten, mir einmal zu schildern, was Ihnen bei diesem Namen alles an Gedanken und Gefühlen durch den Kopf geht? Woran denken Sie, wenn Sie diesen Namen sehen?

	Total	Produktpräferenz				Kaufbereitschaft		Leser Automobil-zeit-schriften	Alter			
		A "Renn-wagen"	B "Luxus-autos"	C "Sport-wagen"	D "Oldtimer"	hoch	niedrig		14-19 Jahre	20-29 Jahre	30-49 Jahre	50-69 Jahre
Basis	300	58	101	38	119	130	170	196	39	78	123	60
Summe der Nennungen	274 % 823	262 % 151	278 % 281	290 % 87	272 % 324	271 % 352	277 % 471	270 % 530	244 % 95	290 % 226	293 % 360	237 % 142
POSITIVE / NEUTRALE KOMMENTARE	74 % 221	76 % 38	71 % 72	80 % 24	73 % 87	73 % 95	74 % 126	74 % 145	82 % 32	73 % 57	67 % 83	82 % 49
Allgemeine Beschreibungen	46 % 137	42 % 21	40 % 40	60 % 18	49 % 58	43 % 56	48 % 81	48 % 95	64 % 25	44 % 34	41 % 51	45 % 27
schöne/ schicke/ tolle Autos/ gutes Design (Schönheit des Autos)	20 % 60	16 % 9	17 % 17	37 % 11	19 % 23	19 % 25	21 % 35	18 % 35	41 % 16	17 % 13	17 % 21	17 % 10
denke an Autos	9 % 28	2 % 1	9 % 9	17 % 5	11 % 13	6 % 8	12 % 20	13 % 25	15 % 6	8 % 6	10 % 12	7 % 4
besonders/ außergewöhnliche Autos/ nicht alltägliche Autos	8 % 23	6 % 3	9 % 9	- 0	9 % 11	8 % 10	8 % 13	11 % 21	5 % 2	12 % 9	6 % 7	8 % 5
Traumautos/ Autos von denen man träumt	6 % 17	4 % 2	4 % 4	7 % 2	8 % 9	6 % 8	5 % 9	7 % 14	3 % 1	6 % 5	6 % 7	7 % 4
Neuvorstellungen/ Prototypen/ Zukunftsvisionen	5 % 16	8 % 4	4 % 4	- 0	7 % 8	4 % 5	6 % 11	5 % 10	8 % 3	5 % 4	5 % 6	5 % 3
faszinierende/ interessante Autos/ Faszination des Autos	5 % 16	4 % 2	5 % 5	10 % 3	5 % 6	8 % 10	4 % 6	4 % 8	3 % 1	3 % 2	7 % 9	7 % 4
große Autos	3 % 8	2 % 1	3 % 3	- 0	3 % 4	3 % 4	2 % 4	3 % 5	3 % 1	3 % 2	4 % 5	- 0
Assoziationen zu "Sport/ Schnell/ Fun"	43 % 129	46 % 23	44 % 44	40 % 12	42 % 50	42 % 54	44 % 75	40 % 78	44 % 17	46 % 36	41 % 50	43 % 26
schnelle Autos/ mit hoher PS- Zahl	18 % 55	16 % 8	17 % 17	17 % 5	21 % 25	14 % 18	22 % 37	17 % 34	23 % 9	18 % 14	15 % 18	23 % 14
Sportwagen/ sportliche Autos	16 % 48	26 % 13	12 % 12	13 % 4	16 % 19	17 % 22	15 % 26	12 % 24	13 % 5	19 % 15	15 % 19	15 % 9
Rennwagen/ Motorsport/ Formel 1	11 % 33	14 % 7	9 % 9	13 % 4	11 % 13	12 % 16	10 % 17	12 % 23	13 % 5	12 % 9	10 % 12	12 % 7
Cabriolets/ 2-Sitzer/ Roadster	7 % 21	4 % 2	13 % 13	7 % 2	3 % 4	8 % 10	6 % 11	5 % 10	- 0	9 % 7	10 % 12	3 % 2

Partner GmbH

Seite 1



Produkttest Autobilder

Frage 1:
Ich habe hier einen Text, der als Name für ein neues Produkt dienen soll. Auch wenn Sie jetzt noch nicht genau wissen, um was für eine Art von Produkt es sich handelt, möchte ich Sie bitten, mir einmal zu schildern, was Ihnen bei diesem Namen alles an Gedanken und Gefühlen durch den Kopf geht?
Woran denken Sie, wenn Sie diesen Namen sehen?

	Total	Produktpräferenz				Kaufbereitschaft		Leser Automobil-zeit-schriften	Alter			
		A "Renn-wagen"	B "Luxus-autos"	C "Sport-wagen"	D "Oldtimer"	hoch	niedrig		14-19 Jahre	20-29 Jahre	30-49 Jahre	50-69 Jahre
Basis	300	50	101	30	119	130	170	196	39	78	123	60
NEGATIVE KOMMENTARE	15 % 44	12 % 6	14 % 14	3 % 1	19 % 23	15 % 19	15 % 25	14 % 27	13 % 5	15 % 12	18 % 22	8 % 5
NEGATIVE AUTO-ASSOZIATIONEN	10 % 29	10 % 5	9 % 9	3 % 1	12 % 14	8 % 11	11 % 18	10 % 19	10 % 4	12 % 9	11 % 13	5 % 3
Umweltbelastung	5 % 16	8 % 4	4 % 4	- 0	7 % 8	4 % 5	6 % 11	5 % 10	5 % 3	5 % 4	5 % 6	5 % 3
billige Autos ("Japaner"/ mindere Qualität)	3 % 8	2 % 1	3 % 3	- 0	3 % 4	3 % 4	2 % 4	3 % 5	3 % 1	3 % 2	4 % 5	- 0
Unfälle	2 % 5	- 0	2 % 2	3 % 1	2 % 2	2 % 3	1 % 2	2 % 4	- 0	4 % 3	2 % 2	- 0
Werkstätten/ kaputte Autos	0 % 1	- 0	1 % 1	- 0	- 0	- 0	1 % 1	- 0	- 0	- 0	1 % 1	- 0
ALLGEMEIN NEGATIV PRODUKT	5 % 15	2 % 1	5 % 5	- 0	8 % 9	6 % 8	4 % 7	4 % 8	3 % 1	4 % 3	7 % 9	3 % 2
allg. Name/ Text gefällt nicht (klingt nicht gut/ ansprechend)	2 % 6	- 0	2 % 2	- 0	3 % 4	2 % 3	2 % 3	2 % 3	3 % 1	1 % 1	2 % 3	2 % 1
klingt oberflächlich/ einfach/ schlicht	2 % 6	- 0	2 % 2	- 0	3 % 4	2 % 2	2 % 4	2 % 4	- 0	3 % 2	2 % 3	2 % 1
klingt platt/ plump/ trivial/ anspruchslos/ protzig/ billig	2 % 6	- 0	2 % 2	- 0	3 % 4	2 % 2	2 % 4	2 % 4	3 % 1	1 % 1	3 % 4	- 0
klingt langweilig/ wenig originell/ uninteressant	1 % 4	- 0	- 0	- 0	3 % 4	1 % 1	2 % 3	1 % 2	- 0	3 % 2	2 % 2	- 0
Autobilder allg. uninteressant	1 % 2	2 % 1	- 0	- 0	1 % 1	2 % 2	- 0	- 0	- 0	- 0	2 % 2	- 0
wenig Informationen/ nur Äußerlichkeiten	0 % 1	- 0	- 0	- 0	1 % 1	- 0	1 % 1	1 % 1	- 0	- 0	1 % 1	- 0

Partner GmbH

Seite 2

SLICESTATISTICS
Summary tables of the type:

```
TABLE = #k by
Mean( v1 )
Mean( v2 )
Mean( v3 )
Mean( v4 )
Mean( v5 )
Mean( v6 )
Mean( v7 )
Mean( v8 )
…..
Mean( v99 )
;
```

can be spread across several pages using the key word `SLICESTATISTICS`. After setting `SLICESTATISTICS = 35;` all the following tables of this type are always divided after 35 such rows.

CALCULATECOLUMN
Syntax:

```
CALCULATECOLUMN = <zielcolumn> [ format "<format>" ] =
<arithmetischer ausdruck>;
```

The notation for the column is: `< <varno> <code> >. <varno>` stands for the tally of the variables in the stub. This notation can now be used as often as desired as a variable in arithmetical output in order to tally the column contents.

For example in order to calculate the difference in the column code 3 of the second variable in the stub to the columns to code 1 and 2 of the first variable the following formulation is required:

```
CALCULATECOLUMN < 2 3 > = < 1 2 > - < 1 1 >;
```

The statements can be as complex as required; for the root of the difference in column 4.5 for example:

```
CALCULATECOLUMN <2 4.5 > FORMAT "#,##" = EXP( LN( < 1 2 > - < 1 1 > )
/ 2 );
```

This is the exponential function of the logarithm of x, divided by 2. This statement means as much as x to the power of 1/2, and that is (almost) the same as the second root. This statement is only defined for values > 0.0. Negative arguments are represented with '-' for `MISSING`.

BTW:
The 4.5 for the code in the above example is no typing error but rather a seldom used feature of ***GESS* tabs**: the codes which are allocated to labels do not have to have whole values.

If the resultant difference is to be presented as a percentage of the first column one can write:

```
CALCULATECOLUMN < 2 6 > = ( < 1 2 > - < 1 1 > ) / < 1 1 > * 100.0;
```

This can only be achieved with a series of limitations: (***GESS* tabs** is not a table calculation program):

- the target column is to the right of the output column in the table,
- `GLOBALCOLMINIMUM` should be set to zero or it must be ensured some other way that the column will not be suppressed as empty,
- the table has only one elementary `CELLELEMENT`: either e.g. `COLUMNPERCENT` or `ABSOLUTE`,
- but not both and no complex `CELLELEMENT` like `ABSCOLPERCENT`,
- only one single figure can be the result printed in the cell,
- decimal numbers must be divided with a decimal point and not a comma,
- the additional column is not allocated a meaningful basis or total,
- the cells are empty, i.e. the table may have to be produced without `ABSROW` or `TOTALROW`.
- tallied columns cannot be used as output columns for further `CALCULATECOLUMN` which would lead to ???.

`TABLE ADD`

`TABLE ADD` can be used to define additional tallies in a table which has already been defined. Complex tables which cannot be defined by simple `TABLE` statements are comfortably and most importantly flexibly handled: de-rotated presentations of rotated questions, tabulation of before and after comparisons, parallel presentation of several variables with the same value labels and more of the same.

The use of `TABLE ADD` is a pre-requisite for the meaningful use of dependent significance tests such as `COLDEPTTEST` and `MEANCOLDEPT`.

If for example three variables are presented next to each other in a cross table, the following can easily be used:

```
DUMMYHEAD = Kopf;
LABELS Kopf =
1 "erste Variable (Var1)"
2 "zweite Variable (Var2)"
3 "dritte Variable (Var3)"
;
TABLE     = Kopf BY Var1;
TABLE ADD = 2 BY Var2;
TABLE ADD = 3 BY Var3;
```

TABLE ADD statements have to have the same structure as the relevant TABLE command: e.g. a MEAN cannot be added up with a PROZENTCOLUMN. In the same way all table elements have to be contained in the original table.

TABLE ADDs can have commands for their own TABSELECT filter, own USEWEIGHT and also their own USECASES commands. Preparation for printing use only the original TABLE, thus all LABELs etc. have to be defined here. All TABLE ADDs are only tally shadows; they are not "evaluated" according to the figures.

When producing such TABLE ADD sequences it can be that a variable is required which does not effect a calculation, i.e. is never counted. In order to define such a "blank" the systems own variable NIL can be used.

As constants are often required for the production of TABLE-ADD tables it is possible to write constants into the TABLE ADD commands where usually a variable would be expected as in the example above. In the actual TABLE statements this makes no sense because numerical constants for example have no VALUELABELS. A variable is often required with a constant content. This variable (with the constant value 1) can be produced most comfortably using the DUMMYHEAD statement.

Some more tips concerning de-rotation: This problem often occurs in the form of formally identical question blocks in a questionnaire being asked for different products more than once and in a different order, and the order is stored in a rotation variable. In our example it is the products A B and C. There are for example three versions of the questionnaire which are identified with the values 1, 2 and 3 in the rotation variable.

The following is valid for the order:

	1st Block	2nd Block	3rd Block
Rotation 1	A	B	C
Rotation 2	C	A	B
Rotation 3	B	C	A

First we look at just one variable form each rotation block with evaluation on a five-step scale:

Die Ausgangsvariablen:

Rotationen:
Rotation 1: a b c
Rotation 2: c a b
Rotation 3: b c a

Abs.	rotation Rotation 1	Rotation 2	Rotation 3
N	87	87	87
Bewertung auf 5er-Skala			
sehr gut	12	12	10
gut	20	25	30
so lala	20	20	15
eher nicht gut	20	18	16
schlecht	15	12	16
eval_2			
sehr gut	9	6	8
gut	22	27	19
so lala	23	17	34
eher nicht gut	27	26	14
schlecht	6	11	12
eval_3			
sehr gut	7	8	12
gut	20	25	17
so lala	20	21	22
eher nicht gut	33	20	22
schlecht	7	13	14

GESS mbH

As long as we are only have to de-rotate one question it is obviously simplest to calculate the "de-rotated" question from the rotation variable and the three rotated questions: If `EvaL_1, Eval_2` and `Eval_3` are the evaluations of the first, second and third product then:

```
if rotation eq 1 then Eval_A = Eval_1;
if rotation eq 1 then Eval_B = Eval_2;
if rotation eq 1 then Eval_C = Eval_3;

if rotation eq 2 then Eval_A = Eval_2;
if rotation eq 2 then Eval_B = Eval_3;
if rotation eq 2 then Eval_C = Eval_1;

if rotation eq 3 then Eval_A = Eval_3;
if rotation eq 3 then Eval_B = Eval_1;
if rotation eq 3 then Eval_C = Eval_2;

copylabels Eval_A Eval_B Eval_C = Eval_1;
```

And we tabulate exactly as in the example above:

```
tabletitle = "Berechnung enrotierter variables";
table =  k    by Eval_A;
citevartext toptext =  rotation;
table add = 2 by Eval_B;
table add = 3 by Eval_C;
```

Berechnung enrotierter Variablen

Rotationen:
Rotation 1: a b c
Rotation 2: c a b
Rotation 3: b c a

Abs.	Produkt A	Produkt B	Produkt C
N	261	261	261
Eval_A			
sehr gut	30	27	27
gut	64	77	64
so lala	59	59	74
eher nicht gut	68	63	65
schlecht	40	35	31

GESS mbH

If however the question blocks contain several questions (e.g. twelve questions) on the products A, B and C it is less time-consuming to de-rotate using a clever definition of the table heads as otherwise all twelve variables would have to be produced in a de-rotated version. Thus three variables are generated rot1, rot2 and rot3 which allow the required table to be generated directly from the original variable:

```
if rotation eq 1 then rot1 = 1;
if rotation eq 1 then rot2 = 2;
if rotation eq 1 then rot3 = 3;

if rotation eq 2 then rot1 = 3;
if rotation eq 2 then rot2 = 1;
if rotation eq 2 then rot3 = 2;

if rotation eq 3 then rot1 = 2;
if rotation eq 3 then rot2 = 3;
if rotation eq 3 then rot3 = 1;
```

To return to our rotation table:
The first three rows deal with rotation 1 (A-B-C) which is the simplest: the first product (rot1) is to appear in the first column (1=A), the second in the second, etc.

In rotation 2 (C-A-B) the first product (rot1) is to be added in the third column (3=C), the second product in the first column (1=A), and the third product (rot3) in the second column.

Thus in rotation 3 (B-C-A) the first product (rot1) belongs in the second column, the second product in the third column and the third product in the first column.

The values in the rotation variable rot1 - rot3 per rotation value are thus identical with the order of the variables in the rows of the rotation matrix: ABC=123, CAB=312, BCA=231.

This produces the following table:

```
tabletitle = "Nach Entrotation via IF";
table      = rot1 by eval_1;
citevartext toptext =  rotation;
table add  = rot2 by eval_2;
table add  = rot3 by eval_3;
```

Nach Entrotation via IF

Rotationen:
Rotation 1: a b c
Rotation 2: c a b
Rotation 3: b c a

Abs.	Entrotierte Produkte A B C		
	Produkt A	**Produkt B**	**Produkt C**
N	**261**	**261**	**261**
Bewertung auf 5er-Skala			
sehr gut	30	27	27
gut	64	77	64
so lala	59	59	74
eher nicht gut	68	63	65
schlecht	40	35	31

GESS mbH

And we see that fortunately the same numerical result is achieved!

And for advanced students! There is an even more elegant solution to building the head variables 1 - 3, namely using `INDEXVAR`:

```
compute k1 = 1;
compute k2 = 2;
compute k3 = 3;

indexvar ro1 = k1 k3 k2 by rotation;
indexvar ro2 = k2 k1 k3 by rotation;
indexvar ro3 = k3 k2 k1 by rotation;
```

For the simple derivation of the order of the k1, k2 and k3 variables: this order corresponds exactly to the third column of the rotation matrix: ACB, BAC, CBA.

```
tabletitle = "Nach Entrotation via Indexvar";
table      = ro1 by eval_1;
citevartext toptext =  rotation;
table add  = ro2 by eval_2;
table add  = ro3 by eval_3;
```

Fortunately this also produces the same result:

Nach Entrotation via Indexvar

Rotationen:
Rotation 1: a b c
Rotation 2: c a b
Rotation 3: b c a

Abs.	Entrotierte Produkte A B C Produkt A	Produkt B	Produkt C
N	261	261	261
Bewertung auf 5er-Skala			
sehr gut	30	27	27
gut	64	77	64
so lala	59	59	74
eher nicht gut	68	63	65
schlecht	40	35	31

GESS mbH

GLOBALTABLEMINIMUM

There was a bug that caused the sub tables in `TABLE ADD` constructs to be individually tested against the `TABLEMINIMUM`. Now only the start table is tested. As the tally results of all the tables (incl. `ADD`) should really be taken the sum of all the `FRAMECELLS` is taken into account for the resultant table. More precisely: all labels in the first variable are added in the stub, then all the labels of the first variable on the y-axis are added. If both n are smaller than the minimum the table is considered empty.

DUMMYHEAD

Syntax:

```
DUMMYHEAD = <name>;
```

Generates a variable with the name `DUMMYHEAD` and the value 1 for a simpler syntax of `TABLE ADD` tables.

HEADERS and TABULATE

HEADERS

Syntax:

```
HEADERS = <tablepart> { / <tablepart> }*n;
```

HEADERS and TABULATE can be used to simplify the request for a standardised table volume:
All TABULATE elements are tabulated against all the heads in HEADERS, whereby always all heads for a variable appear one after another. Where there are more than one head they are divided by a slash (/). Using <tablepart> the usual order of variable names or CELLELEMENT from a variable name is meant. In short: using HEADERS the heads are designated which in a TABLE statement are in front of BY.

TABULATE

Syntax:

```
TABULATE [ INVERSE ] = <tablepart> { / <tablepart> }*n;
```

e.g.:

```
#expand #k1 kopf1 kopf2
#expand #k2 kopf3 kopf4 kopf5
HEADERS = #k1 / #k2;
TABULATE = f1 f2 / f3 MEAN( f3_num ) / f4 / f5 / f6;
```

This command generates ten tables; first kopf1 and kopf2 against f1 and f2, then kopf1 and kopf2 against f3 and the mean of f3_num, etc.

XTAB

XTAB

There is a further possibility of describing cross tables. This second more complicated version makes it easier to tabulate variables next to each other and if necessary to use different weights in one table.

Syntax:

```
XTAB

 [ <taboptions> ]

=

ROWS LOCAL ( <localvarlist> )

{
| <rowdescriptor>
}*n

HEADER
{
| <columndescriptor>
}*n
;

taboptions ::=
[
NAME <tablename>
TITLE <tabletitle>
CELLEEMENTS ( <cellelements> )
FRAMEELEMENTS ( <frameelements> )
```

```
TABLEFORMATS ( <tableformats> )
CONTENTKEY <text>
]

rowdescriptor ::=
[
VARIABLE   <localvarname> [ <sortoptions> [ : <condition> ]
|
OVERCODE   <localvarname> [ <values> ] <labeltext>
|
STATISTICS <text> <cellelement>      ( <localvarname> ) [
<printoptions> ]
]

sortoptions ::= siehe die SORT Optionen des TABLE statements

condition   ::= jede nach GESS Syntax korrekte Bedingung

printoptions ::= [ : USEFONT <fontname> [ SIZE <size> ] | : FORMAT
<formatstring> ]

columndescriptor ::= [ <columnvar> | <singlecolumns> ]

columnvar     ::= VARIABLE <varname> : <setlocalvars> [ : <condition>
]
singlecolumns ::= GROUP              <condition> { | <singlecolumn>
}*n

singlecolumn  ::= <text> : <condition> : <setlocalvars> { :
<columnoption> }*n

columnoption  ::= [ USEWEIGHT <varname> | USEFONT <fontname> [ SIZE
<size> ] ]

setlocalvars  ::= { <localvarname> = <varname> }*n
```

It easiest to follow this with a few examples. This first example does the same as a simple `TABLE` cross table:

```
XTAB
=
ROWS LOCAL ( v1 )
| VARIABLE  v1
HEADER
VARIABLE a1 : v1 = a2
;
```

It is the `XTAB` version of a very simple `TABLE` statement:

```
TABLE = a1 BY a2;
```

The command looks cumbersome mainly because the variable a2 is passed on using an internal construct (a local table variable v1) which already has been allocated after the `ROW` key word. Otherwise it should be noted that as opposed to the `TABLE` syntax, first the rows (`ROWS`) and then the columns (`HEADER`) are defined.

A mean value row for example is added using the key word `STATISTICS`:

```
XTAB
=
ROWS LOCAL ( v1 )
| VARIABLE  v1
| STATISTICS "Mittelwert" MEAN( v1 ) : FORMAT "#,##"
HEADER
VARIABLE a1 : v1 = a2
;
```

This all equates to the TABLE command:

```
TABLE = a1 BY a2 MEAN : DESCRIPTION "Mittelwert" : FORMAT "#,##" ( a2
);
```

The XTAB statement also allows the local definition of overcodes (which do not have to be in the label list):

```
XTAB
=
ROWS LOCAL ( v1 )
| VARIABLE  v1
| STATISTICS "Mittelwert" MEAN( v1 ) : FORMAT "#,##"
| OVERCODE v1 [ 1 : 2 ] "Top Box 1+2"
HEADER
VARIABLE a1 : v1 = a2
;
```

The XTAB statement is of advantage when several variables are to be presented parallel to each other in one table. In order to finish off with an example: the table with five items next to each other which we produced with the TABLE ADD statement (Table 13) can be seen here in the XTAB version. The variables are the same as in the above TABLE statement.

```
XTAB =
ROWS LOCAL ( var  )
| VARIABLE var
| STATISTICS "Mittelwert" MEAN( var )
HEADER
GROUP "5 Items nebeneinander"
COLUMNS
| "Farbe wichtig": 1 EQ 1 : var=item_1
| "Sprach~qualität wichtig" : 1 EQ 1 : var=item_2
| "Gute Ausstattung wichtig" : 1 EQ 1 : var=item_3
| "Marke bevorzugt" : 1 EQ 1 : var=item_4
| "Marke abgelehnt" : 1 EQ 1 : var=item_5
;
```

GESS tabs Beispiel Nr. 20 (XTAB)

Col %	Total	5 Items nebeneinander: Farbe wichtig	Sprach-qualität wichtig	Gute Ausstattung wichtig	Marke bevorzugt	Marke abgelehnt
N	2410	482	482	482	482	482
stimme ganz und gar nicht zu	19.9 %	20.3 %	23.7 %	17.0 %	19.1 %	19.3 %
stimme eher nicht zu	19.2 %	19.7 %	16.8 %	19.9 %	19.1 %	20.3 %
weder/noch	20.0 %	22.2 %	20.3 %	19.9 %	18.3 %	19.3 %
stimme eher zu	19.9 %	18.0 %	19.3 %	21.2 %	21.4 %	19.5 %
stimme ganz und gar zu	21.1 %	19.7 %	19.9 %	22.0 %	22.2 %	21.6 %
Mittelwert	3,0	3,0	3,0	3,1	3,1	3,0

GESS mbH

If we want to add more detail to this table because we want to see the distribution according to sex we need to define five `GROUPS`, and within these five `GROUPS` we differentiate the `COLUMNS` between men and women. As we need to do this five times we use a macro.

```
#macro #Item( &1 &2 )
GROUP "&1"
COLUMNS
| "Männer": geschl eq 1 : var=&2
| "Frauen": geschl eq 2 : var=&2
#endmacro
```

This macro is then called up five times within the table:

```
XTAB
FRAMEELEMENTS ( Absrow ) =
ROWS LOCAL ( var  )
| VARIABLE var
| STATISTICS "Mittelwert" MEAN( var ) : FORMAT "#,##"
HEADER
#Item( "Farbe wichtig"            item_1 )
#Item( "Sprach~qualität wichtig"  item_2 )
#Item( "Gute Ausstattung wichtig" item_3 )
#Item( "Marke bevorzugt"          item_4 )
#Item( "Marke abgelehnt"          item_5 )
;
```

And the result looks like this:

GESS tabs Beispiel Nr. 21 (XTAB)

Col %	Farbe wichtig		Sprachqualität wichtig		Gute Ausstattung wichtig		Marke bevorzugt		Marke abgelehnt	
	Männer	Frauen	Männer	Frauen	Männer	Frauen	Männer	Frauen	Männer	Frauen
N	255	227	255	227	255	227	255	227	255	227
stimme ganz und gar nicht zu	22.4 %	18.1 %	24.3 %	22.9 %	14.9 %	19.4 %	14.9 %	23.8 %	18.0 %	20.7 %
stimme eher nicht zu	18.0 %	21.6 %	15.7 %	18.1 %	23.1 %	16.3 %	17.3 %	21.1 %	21.2 %	19.4 %
weder/noch	24.3 %	19.8 %	18.8 %	22.0 %	17.3 %	22.9 %	20.8 %	15.4 %	17.6 %	21.1 %
stimme eher zu	16.9 %	19.4 %	18.4 %	20.3 %	20.4 %	22.0 %	22.4 %	20.3 %	21.2 %	17.6 %
stimme ganz und gar zu	18.4 %	21.1 %	22.7 %	16.7 %	24.3 %	19.4 %	24.7 %	19.4 %	22.0 %	21.1 %
Mittelwert	2,91	3,04	3,00	2,90	3,16	3,06	3,25	2,90	3,08	2,99

GESS mbH

If instead the five items are required next to each other split into all, men and women then the macro contains five items as COLUMNS:

```
#macro #Items( &1 &2 )
GROUP "&1" &2
COLUMNS
| "Farbe wichtig": 1 EQ 1 : var=item_1
| "Sprach~qualität wichtig" : 1 EQ 1 : var=item_2
| "Gute Ausstattung wichtig" : 1 EQ 1 : var=item_3
| "Marke bevorzugt" : 1 EQ 1 : var=item_4
| "Marke abgelehnt" : 1 EQ 1 : var=item_5
#endmacro
```

And the macro is called up three times in the XTAB statement; for all, for men and for women.

```
XTAB
FRAMEELEMENTS ( Absrow ) =
ROWS LOCAL ( var  )
| VARIABLE var
| STATISTICS "Mittelwert" MEAN( var ) : FORMAT "#,##"
HEADER
#Items( "Total"  "" )
#Items( "Männer"  "geschl eq 1" )
#Items( "Frauen"  "geschl eq 2" )
;
```

GESS tabs Beispiel Nr. 22 (XTAB)

	Total					Männer					Frauen				
Col %	Farbe wichtig	Sprach-qualität wichtig	Gute Ausstattung wichtig	Marke bevorzugt	Marke abgelehnt	Farbe wichtig	Sprach-qualität wichtig	Gute Ausstattung wichtig	Marke bevorzugt	Marke abgelehnt	Farbe wichtig	Sprach-qualität wichtig	Gute Ausstattung wichtig	Marke bevorzugt	Marke abgelehnt
N	482	482	482	482	482	255	255	255	255	255	227	227	227	227	227
stimme ganz und gar nicht zu	20.3 %	23.7 %	17.0 %	19.1 %	19.3 %	22.4 %	24.3 %	14.9 %	14.9 %	16.0 %	18.1 %	22.9 %	19.4 %	23.8 %	20.7 %
stimme eher nicht zu	19.7 %	16.8 %	19.9 %	19.1 %	20.3 %	18.0 %	15.7 %	23.1 %	17.3 %	21.2 %	21.6 %	18.1 %	16.3 %	21.1 %	19.4 %
weder/noch	22.2 %	20.3 %	19.9 %	18.3 %	19.3 %	24.3 %	18.8 %	17.3 %	20.8 %	17.6 %	19.8 %	22.0 %	22.9 %	15.4 %	21.1 %
stimme eher zu	18.0 %	19.3 %	21.2 %	21.4 %	19.5 %	16.9 %	18.4 %	20.4 %	22.4 %	21.2 %	19.4 %	20.3 %	22.0 %	20.3 %	17.6 %
stimme ganz und gar zu	19.7 %	19.9 %	22.0 %	22.2 %	21.6 %	18.4 %	22.7 %	24.3 %	24.7 %	22.0 %	21.1 %	16.7 %	19.4 %	19.4 %	21.1 %
Mittelwert	2,97	2,95	3,11	3,09	3,04	2,91	3,00	3,16	3,25	3,06	3,04	2,96	3,06	2,90	2,99

GESS mbH

If the COLUMNs are to be weighted differently further options can be added using a colon e.g. USEWEIGHT and USEFONT.

Tables where new information (and variables) are continually added per column can also be more simply produced using XTAB.

COMPARE or XCOMPARE

Syntax:

```
COMPAREVAR <name> = <Varlist> ;
...
COMPARE <name ...> [ MEANTEST <varlist> ] ;
or
XCOMPARE <Name ...>;
```

COMPARE produces a comparative table of the frequency distribution of several variables. COMPARE tables look very similar to cross tables the main difference being that a cross table presents and compares the distribution of one variable divided into several categories. A COMPARE table compares several variables in the same population. An example: using a cross table the evaluation of the prime minister could be compared with varying party membership of the interviewees. Using a COMPARE table the evaluation of the prime minister could be compared with the evaluation of other politicians. In the latter we would find the total-n of our survey in each column – except for abstainers etc.

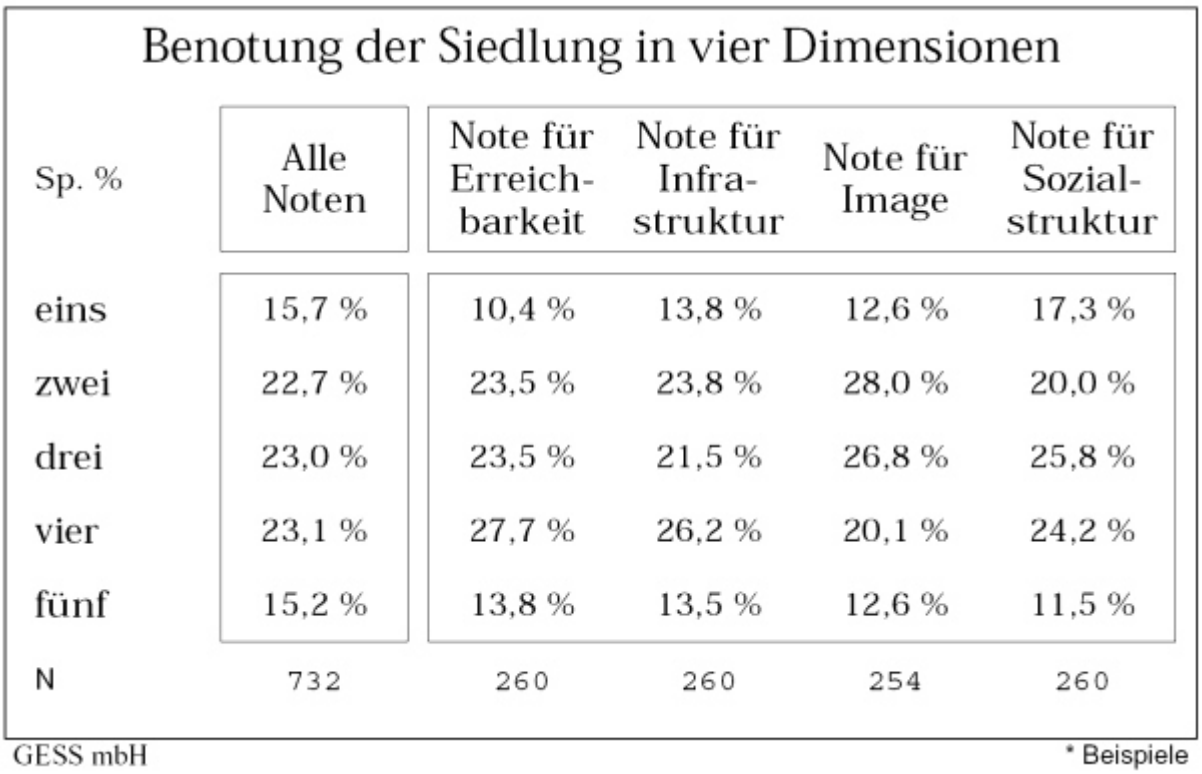
The further development of the TABLE and TABLE ADD statements has however led to many users preferring to use the TABLE statement as it is far more flexible.

If a MEANTEST has been included in the variable list the distribution of this variable is tested against the variance from zero using a t-test. The content is equivalent to that of the t-test for dependent random samples for two variables if the test variable contains the difference between the variables per case.

As of ***GESS* tabs** 2.60 the distributions to be compared can be presented both horizontally and vertically. Vertical presentation is produced using COMPARE, horizontal with XCOMPARE. As of version 2.60 the variables to be compared must be syntactically differently brought together in COMPAREVARs than in the previous version. COMPAREVARs thus behave similarly to VARGROUPs: the VARTITLE of the individual variable become the LABELS of the COMPAREVAR. Labels can however also be allocated to COMPAREVARs (VALUELABELS statement).

An example: a survey was undertaken in a residential area. In this survey the residents were asked to evaluate the neighbourhood: according to access to their area, quality of the infrastructure, image and the social structure of the neighbourhood. The four distributions are to be compared and additionally a total distribution across all evaluations is required. The four basic distributions are contained in v1, v2, v3 and v4.

The result is the following table (the relevant program script follows):



Benotung der Siedlung in vier Dimensionen

Sp. %	Alle Noten	Note für Erreich-barkeit	Note für Infra-struktur	Note für Image	Note für Sozial-struktur
eins	15,7 %	10,4 %	13,8 %	12,6 %	17,3 %
zwei	22,7 %	23,5 %	23,8 %	28,0 %	20,0 %
drei	23,0 %	23,5 %	21,5 %	26,8 %	25,8 %
vier	23,1 %	27,7 %	26,2 %	20,1 %	24,2 %
fünf	15,2 %	13,8 %	13,5 %	12,6 %	11,5 %
N	732	260	260	254	260

GESS mbH

* Beispiele

```
CASESTITLE = N;
VARFAMILY "Alle Noten" = v1 v2 v3 v4;
VARTITLE v1 = "Note for Erreich-bar-keit";
VARTITLE v2 = "Note for Infra-struk-tur";
VARTITLE v3 = "Note for Image";
VARTITLE v4 = "Note for Sozial-struktur";
VALUELABELS v1 v2 v3 v4 "Alle Noten" =
1 eins
2 zwei
3 drei
4 vier
5 fünf;

COMPAREVAR TotalComp = "Alle Noten";   VARTITLE = "";
COMPAREVAR RestComp = v1 v2 v3 v4;        VARTITLE = "";

TABLEFORMAT = +NODESCRIPTION VOVARTITLEBOX;
CELLELEMENTS = COLUMNPERCENT;
FORMAT COLUMNPERCENT = "#,# %";
FRAMEELEMENTS = ABSROW;
FRAMEPOSITION = BOTTOM;
TABLEBASE = NOMINATIONS;
TABLETITLE = "Benotung der Siedlung in vier Dimensionen";
COMPARE TotalComp RestComp;
```

The contents of the `COMPARE` tables are also controlled by the key words `CELLELEMENTS`, `FRAMEELEMENTS` and `FRAMEPOSITION` as with `TABLE` tables. The same test options are also valid. `COMPARE` tables can also contains several cell contents – percentages and absolute values can be presented etc. In order to compare the total evaluation better with the individual evaluations the table is percentaged on the basis of the number of responses (see `TABLEBASE = NOMINATIONS`). This explains the higher n under "all evaluations" in the first column.

COMPARE tables are obviously only useful if all the variables have the same characteristics. The XCOMPARE statement produces the same tables with one small difference in that the distribution is presented horizontally. The same values are only produced if in the XCOMPARE cellelements ROWPERCENT has been set. Thus it can be seen that it is possible to percentage distribution horizontally.

The value labels in the table are first taken from the first variable which appears in the COMPAREVAR which is in this case "All Evaluations". The syntactical request that all variables have to refer to the identical labels has been dropped.

The following TABLEFORMAT is valid for COMPARE tables (XCOMPARE):

MEANINCOMPARE	an additional row (or column) is added which contains the mean of the evaluated variable.

PROFILE

PROFILE defines a mean table with an optional graphical presentation of the mean value. In certain ways PROFILE is to mean as COMPARE is to distribution. PROFILE can also be used to present many variables cohesively. If for example the competence of four political parties regarding eighteen different political matters has been evaluated there are 72 individual variables of which four can be directly compared with each other as obviously it makes more sense to compare the response that the CDU has more economic sense than the SPD as opposed to the CDU having more economic sense than the FDP has environmental sense! Thus it makes sense to be able to sensibly group independent variables.

The end of a group is identified using a slash. All groups must contain the same number of variables, i.e. if the first table row has four variables all other rows must also have four variables.
Example:

```
vartitle v1 =  "Wirtschaft";
vartitle v6 =  "Umwelt";
vartitle v11 = "Soziales";
vartitle v16 = "Innere Sicherheit";
vartitle v21 = "Verkehrspolitik";
vartitle v26 = "Ausländer";
vartitle v31 = "Straßenbau";
vartitle v36 = "Wohnungsbau";
vartitle v41 = "Asylrecht";
vartitle v46 = "Gesundheits-wesen";
vartitle v51 = "Pflege-versicherung";
vartitle v56 = "Städtebau";
vartitle v61 = "Außenpolitik";
vartitle v66 = "Entwicklungs-hilfe";
vartitle v71 = "Kulturpolitik";
vartitle v76 = "Entwicklung in den neuen Ländern";
vartitle v81 = "Finanz-ausgleich";
vartitle v86 = "Industrie-politik";

tabletitle = "Profiltabelle:";
Profile =
 v1  v2  v3  v4 /
 v6  v7  v8  v9 /
v11 v12 v13 v14 /
v16 v17 v18 v19 /
v21 v22 v23 v24 /
v26 v27 v28 v29 /
v31 v32 v33 v34 /
v36 v37 v38 v39 /
v41 v42 v43 v44 /
v46 v47 v48 v49 /
v51 v52 v53 v54 /
v56 v57 v58 v59 /
v61 v62 v63 v64 /
v66 v67 v68 v69 /
v71 v72 v73 v74 /
v76 v77 v78 v79 /
v81 v82 v83 v84 /
v86 v87 v88 v89;
profileheaders = CDU SPD FDP Grüne ;


Toptext =
"
Wahrnehmung der Kompetenz von 4 Parteien in 18 Politikbereichen
```

```
auf 7-stufiger Skala
1 = keine Kompetenz    7 = sehr hohe Kompetenz
";
bottomtext =
"Quelle: Zufallszahlengenerator der Fa. CLARION Software";
```

The resulting table:

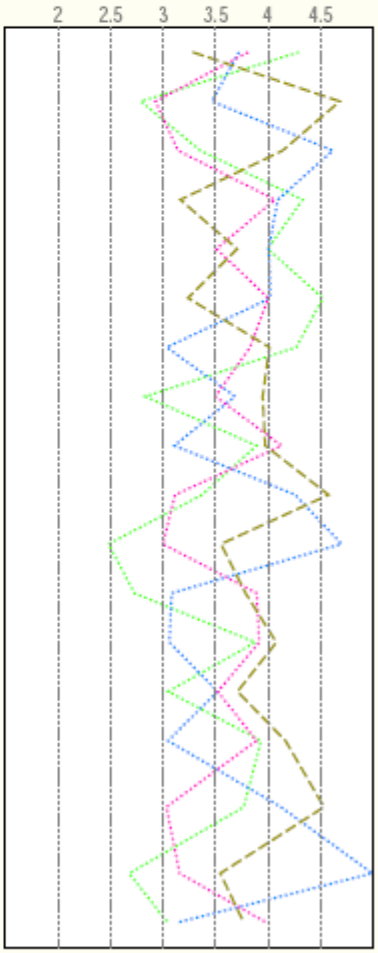
GESS

Profiltabelle: Parteienkompetenz

Wahrnehmung der Kompetenz von 4 Parteien in 18 Politikbereichen

Skala 1 - 7

	CDU 300	SPD 300	FDP 300	B90/ Grüne 300
Wirtschaft	4.3	3.7	3.8	3.3
Umwelt	2.8	3.5	2.9	4.7
Soziales	3.4	4.6	3.1	4.1
Innere Sicherheit	4.3	4.1	4.1	3.2
Verkehrspolitik	4.0	4.0	3.5	3.7
Ausländer	4.5	4.0	4.0	3.2
Straßenbau	4.3	3.0	3.8	4.0
Wohnungsbau	2.8	3.7	3.5	3.9
Asylrecht	3.9	3.1	4.1	4.0
Gesundheitswesen	3.4	4.3	3.1	4.6
Pflegeversicherung	2.5	4.7	3.0	3.6
Städtebau	2.7	3.1	3.9	3.8
Außenpolitik	3.9	3.1	3.9	4.1
Entwicklungshilfe	3.0	3.5	3.5	3.7
Kulturpolitik	3.9	3.0	3.9	4.2
Entwicklung in den neuen Ländern	3.8	4.1	3.0	4.5
Finanzausgleich	2.7	5.0	3.2	3.5
Industriepolitik	3.0	3.1	4.0	3.8



Quelle: TOPSPEED Zufallszahlengenerator

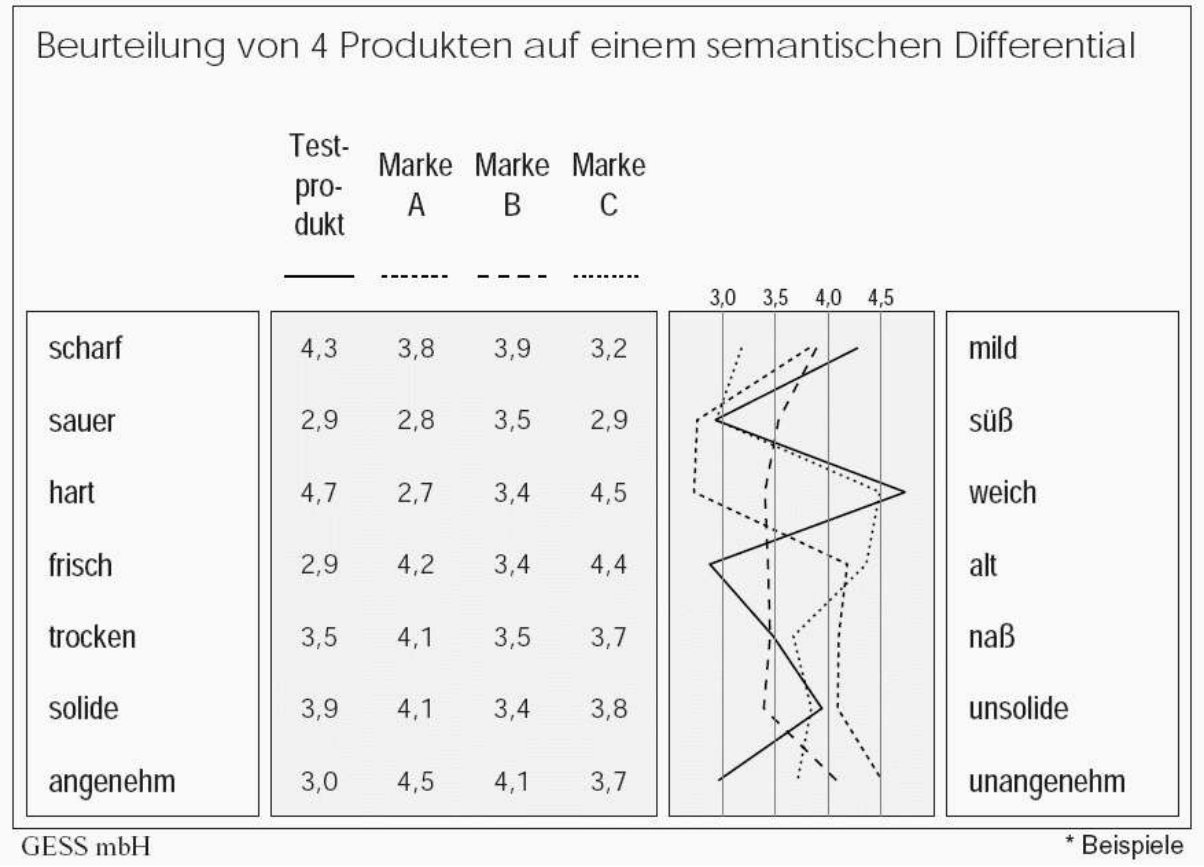
GESS mbH

* Beispiele

The legends for the table columns are taken from the PROFILEHEADERS. The PROFILEHEADERS command always comes directly after the PROFILE command. There have to be the same number of strings as there are variables in a group. The row legends are taken from the VARTITLEs or the variable names of the first variable in each group.

Usually graphics are produced on the right-hand side and a background scale is provided. Both can be switched off using TABLEFORMATs. NOGRID removes the background scale and NOGRAPH removes the whole graphic.

PROFILE tables can have a label on the left or the right. This is useful for the presentation of semantic differences for example. In the following example four products are to be evaluated according to seven semantically opposing pairs. The pairs are to be separated to the left and the right and could look like this:



To achieve this the numerical values for the scale minimum and maximum are written at the end of the variable list in parenthesis in the PROFILE statement. This tells ***GESS* tabs** not to use the variable name or their titles but rather the VALUELABELS allocated to their values. The program script for the above example looks like this:

```
Labels vx1  = 1 "scharf" 7 "mild";
Labels vx5  = 1 "sauer" 7 "süß";
Labels vx9  = 1 "hart" 7 "weich";
Labels vx13 = 1 "frisch" 7 "alt";
Labels vx17 = 1 "trocken" 7 "nass";
Labels vx21 = 1 "solide" 7 "unsolide";
Labels vx25 = 1 "angenehm" 7 "unangenehm";

tabletitle =
"Beurteilung von 4 Produkten auf einem semantischen Differential";
profile =
```

```
vx1 vx2 vx3 vx4 /
vx5 vx6 vx7 vx8 /
vx9 vx10 vx11 vx12 /
vx13 vx14 vx15 vx16 /
vx17 vx18 vx19 vx20 /
vx21 vx22 vx23 vx24 /
vx25 vx26 vx27 vx28 ( 1 7 );
profileheaders = "Test-produkt" "Marke A" "Marke B" "Marke C";
```

PROFILEHEADERS

PROFILEHEADERS is an obligatory command after a PROFILE statement and is used to define the column legends. The test elements can cover more than one row; the same hyphenation rules apply as for VALUELABELS (see above).

PROFILELINES

Syntax:

```
PROFILELINES = { LineDef }*n ;
LineDef ::=   | <number> : { LineQualifier }*n
LineQualifier ::=[ HIDDEN | PATTERN <number> | COLOR <number>
<number> <number> | WIDTH <number> | SYMBOL <number> SYMBOLWIDTH
<number> ]
```

Usually all lines are either black (black-white-definition) or are automatically allocated a colour (Colour-Defaults). With the PROFILELINES statement all lines can be formatted as required.

HIDDEN	the line is not printed
PATTERN	defines the pattern (1-10). (e.g. Pattern 5 = Solid Line, Pattern 1 = Dots)
COLOR	colour is either chosen acc. to HSB (Hue-Saturation-Brightness) or RGB (Red-Green-Blue).
SYMBOL	Symbol number 1 - 5
SYMBOLWIDTH	Symbol size in typographical points
WIDTH	Line width in typographical points.

e.g.:

```
PROFILELINES =
| 1 : HIDDEN
| 2 : WIDTH 2
| 3 : COLOR 0.05 0.78 1.0 PATTERN 5 WIDTH 5
| 5 : HIDDEN
;
```

The lines defined in PROFILELINES remain valid until a new PROFILELINES statement appears.

PROFILESORT

Syntax:

```
PROFILESORT = [ <number> ] [ DESCEND ] ;
```

The PROFILE table is sorted according to the <number> defined in the data column, usually in ascending order; DESCEND defines the descending order. The data column results in the case of a BY table from a code of characteristic. Where there are several variables per row <number> is the first based tally.

PROFILESCALE

Syntax:

```
PROFILESCALE = <start> <end> <increment> ;
```

Usually PROFILE scales the presentation of the graphic itself. It can be desirable to impose a rigid scale e.g. if comparing with other profiles.

In a second syntactical variation the PROFILE statement functions like the TABLE statement: mean can be presented in different sub-categories which have been defined by a further variable. A section of the data about the competence of the CDU from above is to be differentiated in the next example according to the party preferences of the interviewees. The program script looks like this:

```
variable p = title "Bei der letzten Bundestagswahl gewählte Partei"
9;
VALUELABELS =
1 "SPD" USEFONT "Helvetica-Bold" SIZE 13
2 "CDU" USEFONT "AvantGarde-DemiOblique" SIZE 13
3 "F.D.P." USEFONT "Times-Roman" SIZE 12
4 "Die Grü-nen" USEFONT "AvantGarde-Demi" SIZE 10
;

tabletitle = "Profiltabelle: Kompetenz der CDU";
profile =
 v1   v6  v11  v16  v21  v26  v31  v36  v41  v46
 v51  v56  v61  v66  v71  v76  v81  v86  by p;

Toptext =
"
Wahrnehmung der Kompetenz der CDU nach zuletzt gewählter Partei

auf 7-stufiger Skala
1 = keine Kompetenz    7 = sehr hohe Kompetenz

";
bottomtext =
"Quelle: Zufallszahlengenerator der Fa. CLARION Software";
```

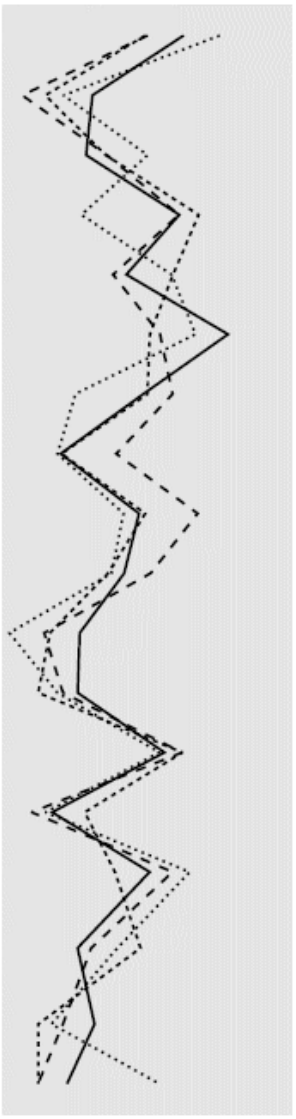
And the table itself looks like this:

Profiltabelle: Kompetenz der CDU

Wahrnehmung der Kompetenz der CDU nach zuletzt gewählter Partei

auf 7-stufiger Skala:
1 = keine Kompetenz 7 = sehr hohe Kompetenz

	Bei der letzten Bundestagswahl gewählte Partei			
	SPD	*CDU*	F.D.P.	Die Grünen
Wirtschaft	4,5	4,0	4,0	5,0
Umwelt	3,2	2,6	2,3	2,8
Soziales	3,1	3,4	3,4	4,0
Innere Sicherheit	4,4	4,7	4,4	3,1
Verkehrspolitik	3,7	4,4	3,5	4,3
Ausländer	5,1	4,0	4,2	4,7
Straßenbau	3,9	4,0	4,3	3,0
Wohnungsbau	2,8	2,8	3,6	2,8
Asylrecht	3,9	4,0	4,7	3,7
Gesundheitswesen	3,7	3,5	4,0	3,5
Pflegeversicherung	3,1	2,6	2,6	2,1
Städtebau	3,0	2,5	2,8	2,8
Außenpolitik	4,2	4,5	4,5	4,2
Entwicklungshilfe	2,7	3,1	2,4	2,6
Kulturpolitik	4,0	3,5	4,3	4,6
Entwicklung in den neuen Ländern	3,0	3,9	3,2	3,6
Finanzausgleich	3,3	2,5	2,8	2,7
Industriepolitik	2,9	2,5	2,5	4,2



Quelle: TOPSPEED Zufallszahlengenerator

GESS mbH

* Beispiele

The following TABLEFORMATs are valid for PROFILE tables:

NOGRID	no scale for the linear graphics in PROFILE tables
NOGRAPH	no graphics for PROFILE tables

TTEST

Syntax:

```
TTEST = [ INDEPENDENT ]
TTESTINDEX <number>
var11 var12 .... var1i /
var21 var22 .... var2i /


varn1 varn2 .... varni /
;
TTESTHEADERS = "titel1" "titel2" .... "titel i";
```

TTEST calculates per row the dependent or independent t-test between the variables in a stated column (TTESTINDEX) and all the remaining variables in the same row. Alle variables up to the "/"are in one row. The number of variables in all rows must be the same.

Dependent:
The dependent t-test between two variables is in terms of content the test of the mean of the difference built per case of these variables against zero. In the column TTESTINDEX (e.g. TTESTINDEX=1) the mean of the variables is printed (in the first row var11). If the dependent t-test (e.g. var11 against var12) is significant the corresponding cell is marked with the significance level.

Independent:
The means are compared taking the scatter into account. The means are always printed in the table. The levels of significance are printed (as shown below). The independent t-test can also be calculated in normal TABLE tables (see MEANTEST).

The significant level can be read off from the larger than and less than symbols:

```
>     <      10%-Niveau
>>    <<     5%-Niveau
>>>   <<<    1%-Niveau
>>>>  <<<<   1-Promille-Niveau
```

Dependent upon TABLEFORMAT SHOWTTMEAN can also print the test value, i.e. the mean of the difference, together with the indication of the significance level. All cells outside of the column defined using TTESTINDEX remain empty if the test for significance is unsuccessful.

For TTEST tables:
SHOWTTMEAN
Prints the mean of test variable additional to the indication of significance levels.

CODEBOOK

Syntax:

```
CODEBOOK [ <VarList> ] ;
```

Produces a simple one-dimensional frequency distributionacross the relevant variables. If a variable has not been explicitly named all the variables are tallied.

Example:

```
CODEBOOK;
```

or

```
CODEBOOK Alter Einkommen;
```

In the following example a general tally across variable v1 with the `VARTITLE` "Partei" is to be printed. Here the fifth characteristic is `MISSING`. Usually `MISSING` values are not printed.

Codebook: einfache Häufigkeitsverteilungen

Partei	Abs.	Sp. %	Kum. %
SPD	85	30,9	30,9
CDU	72	26,2	57,1
F.D.P.	65	23,6	80,7
Die Grünen	53	19,3	100,0
N =	**275**	**275**	**275**

GESS mbH Beispiele

`MISSING` values are printed if the following preset occurs before the table commands:

```
USEMISSING = YES;
```

The same table then looks like this:

Codebook: einfache Häufigkeitsverteilungen

Partei	Abs.	Sp. %	Sp. %	Kum. %
SPD	85	27,4	30,9	27,4
CDU	72	23,2	26,2	50,6
F.D.P.	65	21,0	23,6	71,6
Die Grünen	53	17,1	19,3	88,7
K.A.	35	11,3	Missing	100,0
N =	310	310	275	310

GESS mbH Beispiele

Generally two columns are printed with `COLUMNPERCENT` if the number of the valid responses differs from the number of cases. This can be the case if `MISSING` values are to be printed as in our example. This can also be the case if variables with multiple responses are tallied e.g. `MULTIQ` or `DICHOQ`.

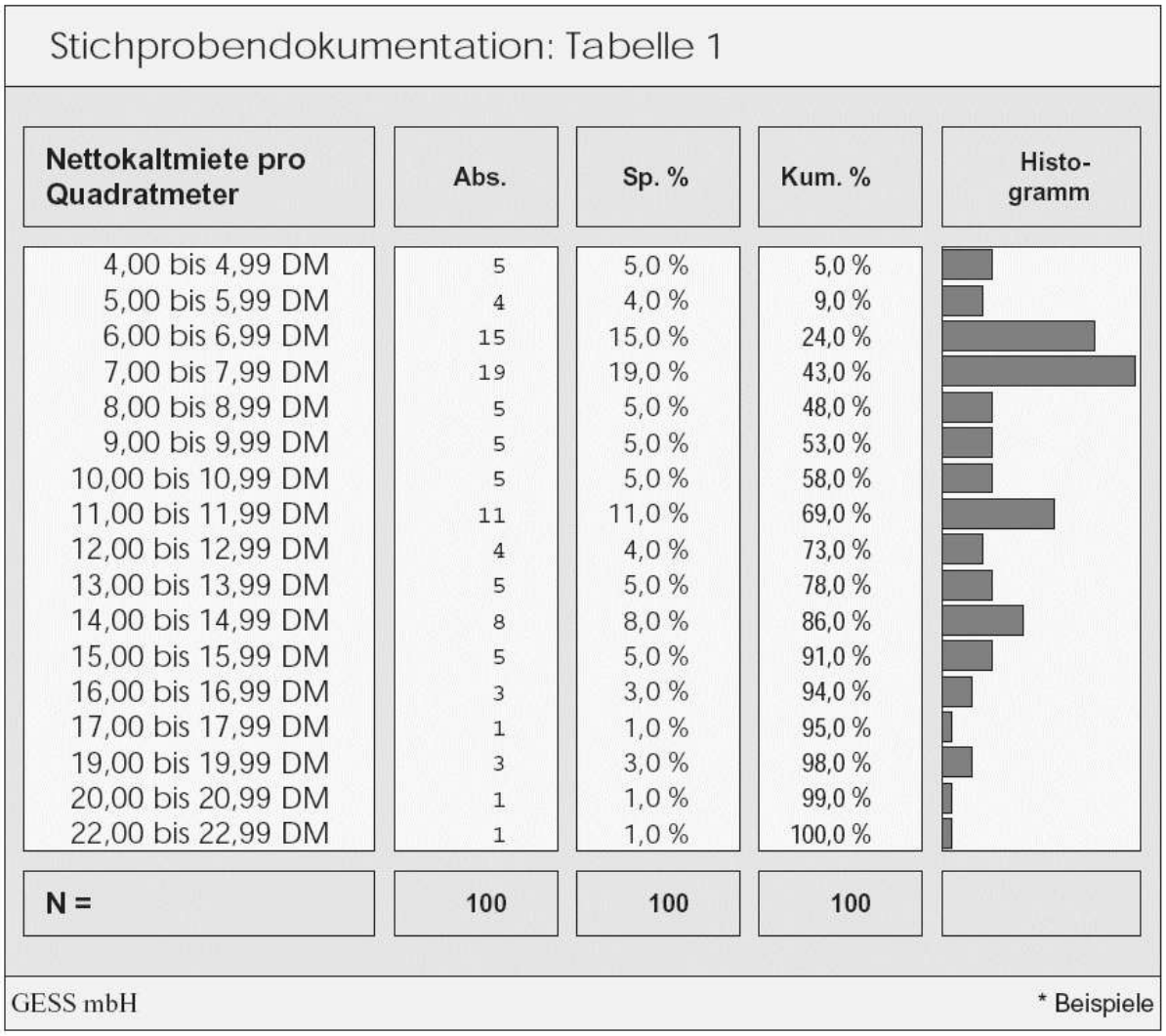
Both tables are produced using the option `TABLEFORMAT = NOGRAPH;` usually a small histogram is produced on the right of the table. The histogram, here left-justified can also be right-justified or centred (see `ALIGN GRAPHBOX`). The `MISSING` values appear in a lighter shade in the histogram.

Codebook: einfache Häufigkeitsverteilungen

Partei	Abs.	Sp. %	Sp. %	Kum. %	Histogramm
SPD	85	27,4	30,9	27,4	
CDU	72	23,2	26,2	50,6	
F.D.P.	65	21,0	23,6	71,6	
Die Grünen	53	17,1	19,3	88,7	
K.A.	35	11,3	Missing	100,0	
N =	310	310	275	310	

GESS mbH Beispiele

`CODEBOOK` also reacts to the `TABLEFORMAT NOBODYBLANKS`; where the rows are produced without gaps. This makes sense when tallying variables with many characteristics. Our example, using a slightly different `TABLEFORMAT` then looks like this:



The following is valid for controlling fonts etc. using CODEBOOK:
DRAWBOX, SHADE and ALIGNMENT: The individual values conform to the DATACELL, the VALUELABELS to LABELS Y. The titles conform to VARTITLE Y (above and below the VALUELABELS) or to VARTITLE X (above and below the values and above the histogram). The histogram itself is governed by GRAPHBOX. LABELSET Y is valid for the box around all VALUELABELS.

Fonts: The fonts are paramount as they are set by USEFONT for the above table elements. The subordinate fonts for ABSOLUTE, COLUMNPERCENT and CUMULATIVE are used for the data columns. Otherwise the standard format is valid.

MAXCODEBOOKLINES
Determines the maximum number of rows per page in a CODEBOOK table.
Preset:

```
MAXCODEBOOKLINES = 50;
```

The following TABLEFORMATs are valid for CODEBOOK tables:

CODEBOOKVALUES
Prints the numerical codes beside the VALUELABELS in CODEBOOK tables. This is particularly useful for controlling the automatic coding which is carried out by ALPHA-VARS.

NOGRAPH no histogram.

COLUMNCOUNT

Syntax:

```
COLUMNCOUNT = <startcolumn> <endcolumn> ;
```

COLUMNCOUNT is not a table in the usual sense. As an 80-Column-Tally it provides a first overview of the data file and is thus more of a technical support or good documentation. In many other systems this presentation is called a HOLECOUNT.

The COLUMNCOUNT command is the only one not to refer to variables but rather directly to the column in the input file. It can therefore only be used in connection with DATAFILE (ASCII-Input) or COLBININFILE (COLBIN-Input). A maximum of 80 columns can be tallied in one statement as that is all that fits on one page.

A data set with columns 1-141 could be counted like this:

```
COLUMNCOUNT = 1 80;
COLUMNCOUNT = 81 141;
```

A two rowed data set could be counted like this:

```
CARD = 1; COLUMNCOUNT = 1 80;
TOPTEXT = "Data set XY: Karte 1";
CARD = 2; COLUMNCOUNT = 1 80;
TOPTEXT = "Data set XY: Karte 2";
```

The "card" in ASCII input is referred to using the key word CARD; in COLBIN COLBININCARD. In Column-Binary-Input the tallied card image is always the direct equivalent of the COLBININFILE. With ASCII-Input the COPYFILE is tallied if a COPYFILE is generated otherwise the card image of the DATAFILE is tallied.

The COLUMNCOUNT can be presented in two different formats if using a line printer (i.e. not Postscript!), see TABLEFORMAT. If using Postscript an additional percentage can be printed:

```
TABLEFORMAT = + PERCENTINLABEL;
```

The following TABLEFORMATs are valid for COLUMNCOUNT tables:

COLCOUNTLINES
Printing of column tallies (COLUMNCOUNT) in a row-orientated format. Usually the results are printed in columns. **(NON-PS)**

An example of an 80-Column-Tally:

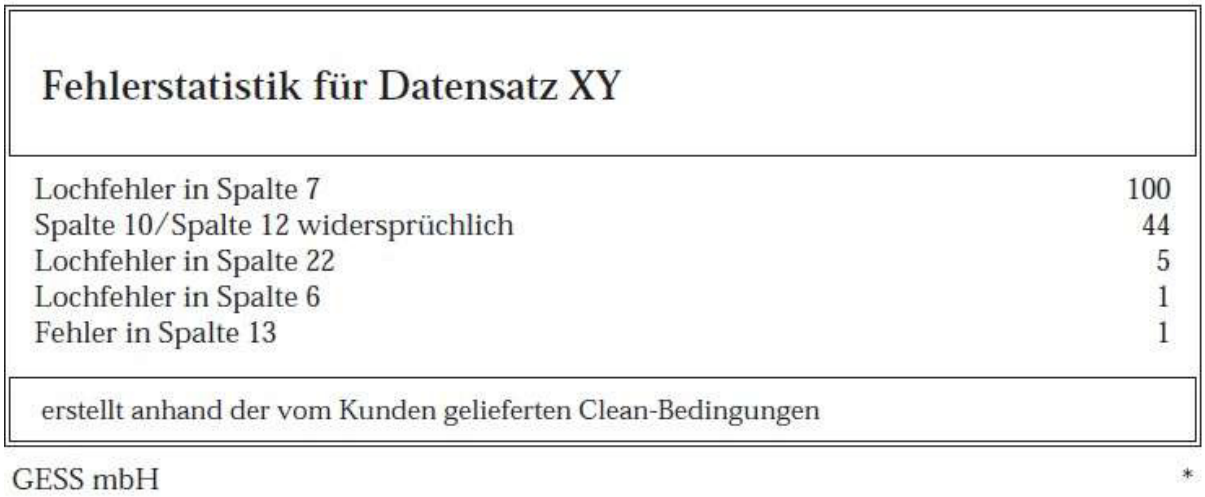
80-Spaltenzählung (ColumnCount) bei Column-Binary-Input

Col.	Blanks	0	1	2	3	4	5	6	7	8	9	X	Y	Single	Multi	Total
1	1591	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	-	-	-	-	-	1591	-	-	-	-	-	-	-	1591	0	1591
3	-	-	-	-	-	-	1591	-	-	-	-	-	-	1591	0	1591
4	-	-	-	-	-	-	-	1591	-	-	-	-	-	1591	0	1591
5	1591	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	257	-	262	257	256	282	275	2	-	-	-	-	-	1334	0	1334
7	19	136	159	163	157	158	160	162	165	154	158	-	-	1572	0	1572
8	-	160	161	157	159	158	157	160	161	158	160	-	-	1591	0	1591
9	-	-	1591	-	-	-	-	-	-	-	-	-	-	1591	0	1591
10	-	-	1591	-	-	-	-	-	-	-	-	-	-	1591	0	1591
11	-	-	-	1591	-	-	-	-	-	-	-	-	-	1591	0	1591
12	-	13	10	115	204	207	138	142	681	26	33	-	22	1591	0	1591
13	-	1312	25	67	62	42	56	21	3	3	-	-	-	1591	0	1591
14	-	7	68	200	225	217	152	128	534	20	21	-	19	1591	0	1591
15	-	-	294	969	328	-	-	-	-	-	-	-	-	1591	0	1591
16	-	-	144	677	292	40	35	364	39	-	-	-	-	1591	0	1591
17	-	-	820	266	505	-	-	-	-	-	-	-	-	1591	0	1591
18	-	-	-	778	813	-	-	-	-	-	-	-	-	1591	0	1591
19	-	-	979	612	-	-	-	-	-	-	-	-	-	1591	0	1591
20	-	-	264	263	250	279	261	274	-	-	-	-	-	1591	0	1591
21	-	-	90	244	368	250	213	426	-	-	-	-	-	1591	0	1591
22	267	-	927	26	12	285	40	34	-	-	-	-	-	1324	0	1324
23	-	-	132	586	728	143	2	-	-	-	-	-	-	1591	0	1591
24	-	-	168	607	639	165	12	-	-	-	-	-	-	1591	0	1591
25	-	-	260	675	516	127	13	-	-	-	-	-	-	1591	0	1591
26	-	-	130	445	495	393	128	-	-	-	-	-	-	1591	0	1591
27	-	-	142	674	586	166	23	-	-	-	-	-	-	1591	0	1591
28	-	-	350	762	415	64	-	-	-	-	-	-	-	1591	0	1591
29	-	5	161	397	587	375	66	-	-	-	-	-	-	1591	0	1591
30	-	15	155	459	568	277	117	-	-	-	-	-	-	1591	0	1591
31	-	-	72	306	553	399	261	-	-	-	-	-	-	1591	0	1591
32	-	-	109	435	544	342	161	-	-	-	-	-	-	1591	0	1591
33	-	15	193	497	568	244	74	-	-	-	-	-	-	1591	0	1591
34	-	-	137	440	474	384	156	-	-	-	-	-	-	1591	0	1591
35	-	-	341	785	192	257	16	-	-	-	-	-	-	1591	0	1591
36	-	-	252	652	607	80	-	-	-	-	-	-	-	1591	0	1591
37	-	-	195	679	640	77	-	-	-	-	-	-	-	1591	0	1591
38	717	-	186	330	186	114	58	-	-	-	-	-	-	874	0	874
39	-	-	1591	-	-	-	-	-	-	-	-	-	-	1591	0	1591
40	452	240	110	338	118	231	62	46	51	6	56	199	81	771	368	1538
41	1121	63	240	21	18	15	5	8	6	-	39	4	90	431	39	509
42	1219	42	5	13	8	241	25	28	14	-	-	3	10	355	17	389
43	-	-	477	609	505	-	-	-	-	-	-	-	-	1591	0	1591
44	-	-	1591	-	-	-	-	-	-	-	-	-	-	1591	0	1591
45	1591	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
46	-	-	1329	262	-	-	-	-	-	-	-	-	-	1591	0	1591
47	1329	15	11	40	6	57	75	11	42	26	-	-	3	238	24	286
48	-	-	159	579	574	236	43	-	-	-	-	-	-	1591	0	1591
49	-	-	146	505	556	308	76	-	-	-	-	-	-	1591	0	1591
50	171	356	525	220	78	268	185	247	181	144	60	98	88	588	832	2450
51	1300	73	10	46	21	74	42	40	-	-	6	-	6	266	25	318
52	1214	19	237	40	5	19	-	-	-	-	42	-	26	366	11	388
53	949	39	108	91	137	48	37	84	40	22	15	95	35	539	103	751
54	1102	22	225	42	13	18	9	52	103	-	-	36	22	442	47	542
55	1209	123	-	-	-	-	-	-	-	-	-	-	259	382	0	382
56	-	-	703	867	21	-	-	-	-	-	-	-	-	1591	0	1591
57	389	61	404	112	260	82	94	143	20	184	161	78	50	787	415	1649
58	1313	45	44	44	14	8	2	-	-	-	111	6	15	267	11	289
59	1322	17	36	139	19	-	-	-	-	-	-	-	70	257	12	281
60	-	-	330	701	359	145	56	-	-	-	-	-	-	1591	0	1591
61	-	-	285	662	386	206	52	-	-	-	-	-	-	1591	0	1591
62	-	-	536	569	308	123	55	-	-	-	-	-	-	1591	0	1591
63	-	-	312	667	426	130	56	-	-	-	-	-	-	1591	0	1591
64	-	-	367	583	381	184	76	-	-	-	-	-	-	1591	0	1591
65	-	698	33	170	431	203	56	-	-	-	-	-	-	1591	0	1591
66	-	-	213	594	488	248	48	-	-	-	-	-	-	1591	0	1591
67	-	-	296	629	484	144	38	-	-	-	-	-	-	1591	0	1591
68	-	-	362	740	378	79	32	-	-	-	-	-	-	1591	0	1591
69	-	-	130	578	606	234	43	-	-	-	-	-	-	1591	0	1591
70	-	-	503	582	281	154	71	-	-	-	-	-	-	1591	0	1591
71	-	-	218	594	421	221	137	-	-	-	-	-	-	1591	0	1591
72	840	-	751	-	-	-	-	-	-	-	-	-	-	751	0	751
73	55	265	86	177	87	84	178	97	142	254	166	-	-	1536	0	1536
74	55	1072	3	-	6	4	192	-	12	23	224	-	-	1536	0	1536
75	-	1391	-	-	-	-	97	-	-	15	88	-	-	1591	0	1591
76	1394	-	197	-	-	-	-	-	-	-	-	-	-	197	0	197
77	34	90	51	89	148	155	316	243	190	200	75	-	-	1557	0	1557
78	34	1019	8	12	3	18	261	2	7	28	199	-	-	1557	0	1557
79	-	1403	-	-	-	-	90	-	-	10	88	-	-	1591	0	1591
80	1317	-	3	68	-	-	155	-	48	-	-	-	-	274	0	274

SUMMARY

SUMMARY

The `IF ... PRINT ...` command in ***GESS* tabs** allows comfortable error searches and documentation. It is however often useful to use statistics for error frequency and `SUMMARY` tables provide just such statistics:



Fehlerstatistik für Datensatz XY

Lochfehler in Spalte 7	100
Spalte 10/Spalte 12 widersprüchlich	44
Lochfehler in Spalte 22	5
Lochfehler in Spalte 6	1
Fehler in Spalte 13	1

erstellt anhand der vom Kunden gelieferten Clean-Bedingungen

GESS mbH *

Syntax:

```
SUMMARY;

Example:
IF Sp.7 LT 1 OR Sp.7 GE 9 PRINT "Lochfehler in Column 7";
.....
TABLEFORMAT = SORTSUMMARYFREQ;
SUMMARY;
TOPTEXT = "
Fehlerstatistik for Data set XY
";
BOTTOMTEXT =
"erstellt anhand der vom Kunden gelieferten Clan-Bedingungen";
```

Using the `TABLEFORMAT SORTSUMMARYFREQ` the error statistic is in descending order according to error frequency. The error text can also be sorted alphabetically: `SORTSUMMARYALPHA`.

The following `TABLEFORMATs` are valid for `SUMMARY` tables:

`SORTSUMMARYFREQ` Summary sorted acc. frequency

`SORTSUMMARYALPHA` Summary sorted alphabetically acc. error text.

TEXTTABLE

Syntax:

```
TEXTTABLE ;
```

TEXTTABLE inserts a further page into the printout. The page fills the printable area (see PAPER, MARGINS), and includes the frame elements and the TABLETITLE. Further text can be simply added using BOTTOMTEXT.

Additional Table Texts

There are many ways of including texts, e.g. questions or explanations, in tables. All table layouts have a "Text box" at the top and bottom which are called TOPTEXT and BOTTOMTEXT.

The following rules are valid for presentation: generally it is attempted to present the texts in the table as they are found in the script, i.e. breaks in the text are adhered to. As it can however look unattractive if the text goes over the edges the rows of text are additionally divided up. Text is not treated as continuous text but rather only the relevant text is divided at the usual places, although this can be altered using:

```
NOWRAPINTEXT = [ YES | NO ] ;
```

to either allow division or not.

Text can be saved either according to variables or tables. Saving according to variables has the advantage that, for example, question text which is to appear in several tables does not need to be written several times. Saving according to tables is however in some cases more flexible. Those texts saved by variable are either adopted automatically by the table (see CITEALLVARS command), or are requested explicitly using CITEVARTEXT.

Note on the positioning of the command texts: TOPTEXT, BOTTOMTEXT and CITEVARTEXT are options for the last table command (TABLE, CODEBOOK, COMPARE or PROFILE) and always refer to the table command which directly precedes it. CITEALLVARS is a preset which refers to all the table commands which follow it.

IGNORETABINTEXT

Syntax:

```
IGNORETABINTEXT = [ yes | no ];
```

Tabulators can be important carriers of information on indentation in the text for Postscript output and often only refer to indents which aid legibility for the reader. The interpretation of these tabulators can now be influenced by IGNORETABINTEXT. If set to YES they are ignored.

HELPTEXT

Syntax:

```
HELPTEXT <VarList> = "text text ";
```

Defines a help text which can be called up during CATI/CAPI or Data Entry (F1 = help button).

VARTEXT

Syntax:

```
VARTEXT <VarList> = "text text ";
```

This command can be used to allocate questions or explanatory texts to variables. After a TABLE statement these texts can be called up for printing in the lower TEXTBOX using the CITEVARTEXT command. The CITEALLVARS command ensures that the texts for all variables in a TABLE table are printed.

Example:

```
VARTEXT Geburtslaune ="
Quelle: Säuglingsbefragung Klinikum Aachen September 1992
Frage:  Wie fühlen Sie sich jetzt, direkt nach der Geburt?
";
```

VARTEXT, TOPTEXT etc. can also be used for the table printout with "@Varname construct. The system then attempts to discover the last value of the variable "varname" and to integrate the text value in the text. The conditions for this are twofold:

- the variable must not be deleted in the tabulation mode; this can be ensured by using:
  STATIC <varname> = YES;

- the variable search tree must still be present after compilation. this can be ensured by using:
  REDEFINEVARS = YES;

COPYTEXT

Syntax:

```
COPYTEXT <varlist> = <variable>;
```

Parallel to COPYTITLE a saved VARTEXT can be repeatedly reused.

CITEVARTEXT

Syntax:

```
CITEVARTEXT [ TOPTEXT | BOTTOMTEXT ] = <Varlist> ;
```

Example:

```
TABLE = einkommen by alter;
CITEVARTEXT BOTTOMTEXT = einkommen;
```

Using this command order only the text of the question "einkommen" is printed in the lower TEXTBOX, the question text "alter" is ignored (provided the variable "einkommen" has had a text defined using VARTEXT)

CITEALLVARS

Syntax:

```
CITEALLVARS = [ TOPTEXT | BOTTOMTEXT | NO ] { XVALID | YVALID };
```

Example:

```
CITEALLVARS = BOTTOMTEXT;
```

All the variables going into a table are examined as to whether a VARTEXT has been defined and where this is the case, the relevant text is printed at the relevant place. If only variable texts from the stub are to be used then XVALID can be used (only variables on the X-Axis are valid). If only the variables from the pre-column are required then YVALID can be used.

Example:

```
CITEALLVARS = TOPTEXT XVALID;
```

The CITEALLVARS command is a preset for all the following tables. It is saved in the system file and thus also affects the tabulation using ***GESS*** menu. It works as though all the following tables each contain a CITEVARTEXT command as an option in which those variables are tallied which have a VARTEXT.

BOTTOMTEXT

This is an alternative method for defining text at the end of a table. BOTTOMTEXT has in contrast to the CITEVARTEXT command which refers to variables, the text to be printed as its argument. If a BOTTOMTEXT is defined any additional CITEVARTEXT or CITEALLVARS commands for the BOTTOMTEXT are ignored. Maximum text length: 1500 characters.

TOPTEXT

This defines a table specific text box in the upper main body of the table. TOPTEXT is as BOTTOMTEXT an option for TABLE, COMPARE, PROFILE or CODEBOOK command and is directly behind a table command. It always has the same priority rules. Maximum text length: infinity.
Example:

```
TABSELECT = einkommen GE 2000;
TABLE = einkommen BY alter;
TOPTEXT = "
nur Personen mit Einkommen von mindestens 2000 €
";
```

In TOPTEXT and BOTTOMTEXT boxes rows that are too long are now divided (2009/12) unless TEXTWRAP has been set in which case the text remains continuous text.

IV Definition of the Input and Output Files

***GESS* tabs** can read the data to be processed either from an ASCII-file (key word DATAFILE, historically also INFILE), from a Dbase-file (key word DBASEIN), from a file in COLumn-BINary-Format (COLBININFILE), from a ***GESS* tabs** system file (SYSTEMIN), by means of SPSSINFILE from a SPSS-system file (*.sav), or even from a CSV-file (DELIMITEDIN). It is not possible to use SYSTEMIN, COLBININFILE, SPSSINFILE, DELIMITEDIN or DATAFILE simultaneously. The printout of system files is achieved using SYSTEMOUT. SYSTEMIN and SYSTEMOUT can only occur once per run. Additional information can be input using associated data files (ASSOCIN).

The results in table form are usually a printout. The key word for printing is PRINTFILE.

The results of data conversion or a factorial weighting can be saved in a new ASCII-file. It is easiest to do this using COPYFILE. In COPYFILE all variables keep the same position as in the DATAFILE. COPYFILE thus only functions in connection with DATAFILE. The COPYFILE contains exactly those variables for which there is an input definition; i.e. variables which for example have been formed with COMPUTE are only present in the COPYFILE if they have an input definition.

More flexible but also more work is the definition of an explicit ASCIIOUTFILE. In an ASCIIOUTFILE only those variables are carried over which have been explicitly named in an ASCIIOUT statement.

COLBINOUTFILE functions in the same way: only those variables (or only the variable characteristics) are carried over which have a COLBINOUT definition.

(See also the explanation for LISTFILE in "XI. Miscellaneous Commands")

DATAFILE or INFILE
Statement of ASCII input files.
Example:

```
DATAFILE = FILE0001.DAT;
```

There can be any number of DATAFILE statements. All input files are read and processed in the order of appearance. (This option is especially for the processing of waves etc. In order to be able to read data sets with different compositions in a system file there are special mechanisms for redefining variables and recoding files. See Chapter XII For Advanced Students)
Example:

```
DATAFILE = WELLE.1;
DATAFILE = WELLE.2;
DATAFILE = WELLE.3;
etc.
```

In the DATAFILE statement OS wild cards can be used.
Example:

```
DATAFILE = "WELLE.*";
```

The chosen directory is searched for all files with the relevant name, the list of file names is sorted alphabetically and all the file names are stored as input files. Therefore if exactly WELLE.1, WELLE.2 and WELLE.3 were in the present directory the wild card phrasing would be functionally the same as the phrasing above. (NB: Take care with .BAK-Files). The individual file number can be called up from the system variable system file number. The tally is based on 1, i.e. the first file is number 1. The file names are allocated to the Variable System File No as Labels.

The input data from ASCII files are expected in "rows" of any length; each case can contain a maximum of 500 such rows.
GESS input, ***GESS*** capi:

DATAFILE is interpreted as being the output file for the data. If a "*" is set as the extension, e.g. DATAFILE = "mydata.*"; then it is expected that the environment variable ***GESS*** will be used. The content of this environment variable replaces the * wild card character and serves to identify different input stations.

OPENQFILE

If open questions are to be used at least one OPENQFILE must be defined. ***GESS* tabs** reads all OPENQFILEs and creates a data bank which allocates which values belong to which case numbers. The OPENQFILE statement has the same syntax as the DATAFILE statement. (The tips for ***GESS*** input and ***GESS*** questionnaire programs are valid where relevant.)

Usually the CASENUMBER is used as the key in OPENQFILE but any variables can be used:

```
KEY OPENQFILE = <varname> ;
```

This variable must be nuclear but can also be ALPHA .

OPENASALPHA

Usually the results of the coding are taken from the OPENQFILEs but the texts can also be used verbatim which is achieved using: OPENASALPHA <varlist> = [YES | NO];

ASSOCFILE

Syntax:

```
ASSOCFILE [ BIG | DBASEIN ] = <filename> KEY <varname> <startcol>
<len> ;
```

Generally associated data sets can be referred to using ASSOCFILE as long as there are numerical keys of the same name in the DATAFILE and in each ASSOCFILE. ASSOCFILEs are one row column fixed ASCII data sets whose content has been made available using ASSOCVAR statements. <startcol> and <len> denote position and length of the key field in the ASSOCFILE. KEY <varname> denotes a variable in the DATAFILE which holds the key information for every case. Identical keys can appear in the DATAFILE as well as in the ASSOCFILE.

In the language of multi-level analysis both superior and subordinate levels can be introduced. In the language of relational database systems 1:1, 1:n, n:1 as well as n:m transformations can be introduced.

Usually the information is stored form the ASSOCFILE in RAM before the DATAFILE is read. This naturally has the advantage of the information being found swiftly using the keys. Using the key word BIG before the allocation symbol instructs the system to read the information from the ASSOCFILE while it is reading the DATAFILE. Before reading the DATAFILE an index is created. This variant is slower but far more sensitive when processing large background files e.g. for panels etc.
(As of Version 2.91i)

As of Version 3.0 BIG should not be required when using Windows or Linux (gt or gtc) if enough RAM is available. Also as of Version 3.0 the key variable can also be ALPHA.

A DBASE file can also serve as the source if the key DBASEIN is given as qualification. All the fields of the DBASE file are then automatically defined as ASSOCCVAR.

ASSOCVAR

Syntax:

```
ASSOCVAR <varname> = [ ALPHA] <startcol> [ <len> [ <width> ] ] ;
```

Every ASSOCVAR statement produces a variable called <varname>. This variable has the internal type OPENVAR and can contain several values (1:n or n:m-mapping). Using <startcol> and <len> simple variables are addressed; with <width> VARFAMILY variables can be addressed. The contents are treated as ALPHA variables if they have been requested using ALPHA.

ASSOCVAR ALPHA has a special feature: if the contents are to be output in an ASCIIOUTFILE the contents must first be carried over to a "normal" ALPHA variable using COMPUTE ALPHA.
Example:

```
COMPUTE ALPHA normal = AssocAlphaVar;
ASCIIOUT normal = 10 20;
```

ASSOCEND

Syntax:

```
ASSCOCEND <filename> ;
```

Every ASSOCFILE statement MUST end using a related ASSOCEND statement. A nesting of ASSOCFILEs is not allowed but after ASSOCEND there can be another ASSOCFILE. <filename> is to be written exactly as in the ASSOCFILE statement.

Here is a cohesive example. This is written in the file FAMILY.TAB:

```
datafile = "fam.*";

variable famnr = 1 5;
variable kinderzahl = *;
variable ort = * labels
1 hamburg
2 kiel
;

#ifdef tabtask
  assocfile = kin.000 key famnr 1 5;
  assocvar Alter = 7 2;
  evalfamvalonce schulort = no;
  assocend kin.000;
  table = famnr by Alter;
  table = ort by Alter;
#end

end;
```

This was produced with the ASSOCFILE parallel to the TAB-File: (this is obviously not obligatory)

```
datafile = "kin.*";

variable famnr = 1 5;
variable kindernr = *;
variable Alter = 7 2 ;

end;
```

Here a simple 1:n transformation is performed: each family can have more than one child; the relationship is produced using the key <famnr>. As the same <famnr> can appear in several records of the children's file the age variable can also contain more than one response. It is easiest here to present this in OPEN variables. If RECODES etc. are necessary then the values in relevant family variables must be copied.

EVALFAMVALONCE in the first TAB file ensures that several children in one family can have the same age.

SPSSINFILE
Syntax:

```
SPSSINFILE = <filename>;
```

All variables in the SAV-files are produced as ***GESS* tabs** variables with the name allocated to them and VARLABEL and VALUELABEL inherited. All SPSS variables are taken in ***GESS* tabs** as nuclear variables; multi response variables may have to be put together by hand in the script. Numerical and CHAR variables are supported, the latter are made into ALPHA variables. Prerequisite for the use of SPSSINFILE statements is that the DLL (SPSSIO32.DLL) from the installation CD for SPSS can be found by the system. This DLL must therefore either be in the "usual" place (\WINNT\SYSTEM32) or in the same directory as ***GESS* tabs**. (newer versions of SPSS need to have additional DLLs installed.)

Any number of SPSSINFILE statements can appear in a ***GESS* tabs** run; these files are then processed one after the other. The system comments on possible errors: variables which do not appear in all files and VALUELABELS that differ in different files.

Upgrade 07/2009: usually SPSSIO32.DLL delivers only shortened VARLABEL texts in order to be compatible with older versions. ***GESS* tabs** now requests explicitly the adoption of the complete text.

SPSSSOUTFILE
Syntax:

```
SPSSSOUTFILE = <filename>;
```

Stores all variables in a SPSS-sav-file. NOSPSS is taken into account parallel to generating SPSS syntax description. With MULTIQ und DICHOQ the constructing nuclear variables are carried over.

If a VARTEXT is present it is stored in VARLAB otherwise the VARTITLE is stored if present.

MISSING is carried over as well as all labels apart from those between 65001 and 65500. (internal synthetic codes reserved for system use (AUTOANSWER and OVERVODE).

LABELSTOTITLE (SPSSINFILE**)**
Syntax:

```
LABELSTOTITLE <labelcode> = <varlist>;
```

As it obviously often occurs that the interesting texts from 0/1 variables in the SAV files are not in the VARLABEL but in a label this statement can save the editor some work.
Example:

```
LABELSTOTITLE 1 = dich1 TO dich20;
VARGROUP dich = ( dich1 TO dich20 ) EQ 1;
```

SPSSVARLABTOTEXT (SPSSINFILE)
Syntax:

```
SPSSVARLABTOTEXT = [ YES | NO ];
```

SPSS only recognises VARLABEL and not variable texts but whole texts can be sunk into newer versions. If this VARLABEL is to be used as a question text in ***GESS* tabs** this can be achieved with this option.

DELIMITEDIN

Syntax:

```
DELIMITEDIN [ delim ] = <filename>;
delim ::= '<char>' | "<char>" | number [ 1..255 ]
```

Allows the reading of data rows which contain variable values separated by a delimiter, as in CSV files for example. Usually a semi colon is expected but other delimiters before the allocation symbols can be defined. The variable name is expected in the first row also separated by a delimiter.

All variables are appointed to the name printed in the first row of the file where this is possible (variable names can only be defined once and must begin with a letter etc). The variables are all nuclear numerical. Valuelabels, Texts etc. are naturally undefined and can or must be allocated in the script.

Historically DELIMITEDIN is restricted to numerical variables but as of 10/2009 the convention is valid that variables are treated as ALPHA if their name in the head row of the DELIMITEDIN file begins with a @ symbol.

DBASEIN

Syntax:

```
DBASEIN = <filename> ;
```

A dBase -DBF file is read as the input file. The DBASEIN statement evaluates the header of the dBase-File and produces variables with the column names. NUMERIC columns are processed in ***GESS* tabs** as numbers, all other variables are read as ALPHA variables. MEMO fields are ignored.

In this case one should not produce any more own variables from the input (SINGLEQ, VARNAME etc.)

COPYFILE

The output of processed and perhaps altered data sets to an ASCII file. With exception of RECODEs, COMPUTEs etc. (see below) the content of the COPYFILE is identical to that of the DATAFILE. (for historical reasons the key word OUTFILE is accepted as a synonym.)

(see also ASCIIOUT ALL;)

ASCIIOUTFILE

Syntax:

```
ASCIIOUTFILE [ DELIMITED [ ASCIIOUT ] ] = <filename>;
```

Output of data in ASCII format. The ASCIIOUTFILE contains all the variables that have ASCIIOUT (see below). Usually the ASCIIOUTFILE is generated per column but for the transfer of the data to external programs such as EXCEL outputs separated with delimiters can be written without format. This is achieved using the key word DELIMITED. In this case normally a copy of the processed data is generated, i.e. no ASCIIOUT statements need be written. If only certain variables are to be included in the DELIMITED ASCIIOUTFILE then an ASCIIOUT can be written afterwards.

Normally the internal ***GESS* tabs** order of the variables is kept during output into a DELIMITED file. Due to special request this has been expanded. Should:

```
ASCIIOUTFILE DELIMITED VARIABLES = <filename>;
```

be written instead of:

```
ASCIIOUTFILE DELIMITED ASCIIOUT  = <filename>;
```

then the variables in the output file are no longer stored in the internal order but in the order they are named in the ASCIIOUT statement.

NOASCIIEXTENSION
Normally ASCII data sets which have been produced by ***GESS* tabs** are finished with a right-justified *.Should this not occur it can be achieved with a switch.

SYSTEMIN
Output of a system fileto be read. It has to refer to a valid name in the system software. System files are generated with the statement SYSTEMOUT. DATAFILE, COLBININFILE and SYSTEMIN statements can not be used together in a ***GESS* tabs** run. The suffix (.TS) is generated automatically.

SYSTEMOUT
Syntax:

```
SYSTEMOUT = <filename> [ [ KEEPVARS | DELETEVARS ] <varlist> ] ;
```

Output of an OS file name for a system file which is to be generated. SYSTEMOUT statements can be used together with ASCIIOUTFILE, COPYFILE or COLBINOUTFILE statements. In the system file the variables, the VALUELABELS, page definitions, modified standard texts etc. as well as the cases are stored. Regarding suffix – see above.

KEEPVARS or DELETEVARS lead to only some of the variables being carried over to the system file. The variable lists can contain the TO-convention.

COLBININFILE
Syntax:

```
COLBININFILE = <filename>;
```

Input file for data in COLumn-BINary-Format. (here the punch-card with its 12 punching positions celebrates its happy virtual come-back!)

COLBININCOLS
Syntax:

```
COLBININCOLS = <value>;
```

COLBINOUTCOLS
Syntax:

```
COLBINOUTCOLS = <value>;
```

Originally Column-Binary rows were always 80 "columns" long. Physically this is equivalent to a series of 80 16-Bit-Words (thus 160 Bytes) in each of which twelve possible punches of a punch card (that's exactly where they came from) are stored, four bits per word remain unused. Following there are usually two bytes with CR and LF which limit the relevant record to a "row". Physically a COLBIN record is thus usually 162, in some cases 160 bytes long.

The limit on the length fell in connection with the QUANTUM rows; here all single punches were written as 1 byte (as in an ASCII-File), all multi punches were replaced by a *, and directly after the row there was a delimiter (HEX 7F) and then there were two bytes each with COLBIN in the order of the values replaced by an asterisk. These rows were soon longer than 80 characters and in these cases ***GESS* tabs** automatically adapted the length as it recognised the end of a row due to the CR/LF sequence.

In the meantime there were however also data sets which could have any length in the original binary COLBIN format. Theses data sets have arbitrary record lengths which cannot be extracted from the binary data. In order to adapt the reading process the COLBININCOLS statement is used.

A data set with a 900 byte physical string length would thus be modified using:

```
COLBININCOLS = 450;
```

(two bytes per COLBIN column). Maximum value: 800.

COLBININSWAPPED
Syntax:

```
COLBININSWAPPED = [ YES | NO ];
```

COLBIN data can be "swapped", i.e. the order of HIGH and LOW bytes does not conform to the INTEL order for the 8086 processor family. In this case ***GESS* tabs** can cause the bytes within the 16-Bit-Words to be swapped before processing.

COLBINOUTFILE
Output of data in COLumn-BINary-format. See COLBIN-Data above.

COLBINOUTSWAPPED
Syntax:

```
COLBINOUTSWAPPED = [ YES | NO ];
```

COLBIN-Data sometimes also needs to be "swapped" for output, i.e. the order of HIGH and LOW bytes has to be reversed. In this case ***GESS* tabs** can cause the bytes within the 16-Bit-Words to be reversed before storage.

COLBINCRLF
Syntax:

```
COLBINCRLF = [ YES | NO ];
```

In some installations COLBIN data sets are laid down as 160 bytes in others as "rows" finished off with a CR/LF sequence. In the latter the COLBIN records are 162 bytes long in both the input and output.

COLBINFORMAT
Syntax:

```
COLBINFORNAT = <Colbinformatname>;
```

Necessary adjustments to special Column-Binary-Formats are achieved using the COLBINFORMAT statement. All presets are valid for input as well as for output. Permissible COLBINFORMAT names:

ELDAS	private format of the "Institut for angewandte Sozialwissenschaften in Bad Godesberg" (infas).
QUANTUM	QUANTUM data format (®). Data format comprises ASCII-Columns. If there is an X or Y or multi-punch in a column a * is set at the relevant position. Directly after the last occupied position there is a special character (ASCII 127). Then there are two bytes for each asterisk which form a COLBIN-Word. In order to make these bytes into printable ASCII-characters each uppermost bit is set, thus the byte values are between 64 (empty COLBIN half column) and 127 (all 4 Bits in the COLBIN half column are set).

QUANTUMINCHARS
Syntax:

```
QUANTUMINCHARS = <filename>;
```

Apart from the method mentioned above of marking multi-punches with an asterisk and placing bit patterns at the end of rows there is a further possibility for torturing software developers: certain bit patterns (not all of them, but some of them) can be coded as letters of the alphabet or in special characters. Thus the % symbol is represented by the codes 0, 4 and 8, the capital S as 0 and 2, etc. The text file for this is QUANTUMCODES.ini, which ***GESS* tabs** can provide. This can be used to include the interpretations.

PRINTFILE

Syntax:

```
PRINTFILE <Druckername> = <FileName>;
```

Printout of the output medium or output file for the tables generated.

The print name is either "POSTSCRIPT" (alias "PS") or "ENCAPSULATED" (alias "EPS") or an arbitrarily chosen name for the printer used, e.g. "Laserjet III". The key words POSTSCRIPT or ENCAPSULATED request output from a Postscript interpreter.

Postscript printers process internally completely differently to printers which print per lines. Therefore there are unfortunately a few differences in the ***GESS* tabs** program language. As far as possible ***GESS* tabs** programs are processed as usual; non-valid commands are ignored. In principle it is true that the PRINTFILE command should be at the beginning of the instruction file so that further instructions that may depend on the printer can be interpreted correctly. This is also true for VALUELABELS commands for example VARIABLE commands with a LABELS clause. The USEFONT statement for the standard font should come directly after these (see USEFONT).

If using "EPS" or "ENCAPSULATED" no cohesive Printfile is produced. Instead an individual file is generated for each table in which the table is stored as an encapsulated Postscript graphic. Encapsulated files cannot be sent directly to a printer but they can be read by a Postscript compatible word processing system and used as an illustration. All encapsulated files are generated with the suffix .EPS. Other suffixes are ignored. All EPS files in a program run are numbered. If there is a number at the end of the EPS file name then this is used as the format picture, i.e. the number of numerical positions is carried over to the *.EPS name. The tally of the *.EPS files always begins with the number 1.

Example:

```
PRINTFILE EPS = TEST00;
```

Assuming that four tables are generated the EPS files will be named TEST01.EPS, TEST02.EPS, TEST03.EPS and TEST04.EPS.

The PRINTFILE statement supports conditional compilation in that internally either the name PS or NON-PS is generated as DEFINE. Directly afterwards the pre-processor #IFDEF or #IFNDEF can be used to prompt which printer is to be used.

V Variables in *GESS* tabs

In ***GESS* tabs** the concept "variable" is often synonymous with the concept "question". From the viewpoint of the programmer who is programming a questionnaire for CATI or CAPI it is all about presenting a series of questions; the tabulator who comes afterwards sees in the same material rather a single or a multi variable whose contents are to be tallied and then visualised.

In order to be able to write a question in a data set or read a variable from an ASCII or COLBIN data set the system must be told the structure of the data set. Often one-row cases will be enough as ***GESS* tabs** can process rows of any length. The data can however be spread over several rows per case (max. 800).

Apart from such questions/variables which are written in code in a data set or read from the data set there are also those which refer to the value set from the coding of an open question. ***GESS* tabs** uses its own format for OPENQFILES which is provided for data entry or questionnaire programs. The coding is directly allocated to the open questions in the OPENQFILEs and ***GESS* tabs** processes the codes without them having to be carried over to the main data.

As well as questions or variable names each variable can also be allocated a variable title which is printed instead of the variable name in the table. Key word: VARTITLE.

Using VALUELABELS numerical values are allocated to texts which describe their characteristics. These label texts can also contain separation commands in order to divide texts in tables. Using the key words USELABELS, COPYLABELS or AS and COPY further variables can be referred to using VALUELABELS which have already been defined. This saves time and has the advantage that any corrections need not be undertaken individually. In the same way COPYTITLE can inherit VARTITLE and COPYTEXT VARTEXT.

All these specifications can be combined in the SINGLEQ statement (synonymous with the VARIABLE statement). The VARIABLEVARS and COLUMNVARS statements simplify the writing of many similar variables. As well as these statements there is also the VARNAME statement for generating variables.

The DICHOQ and the MULTIQ statement is for defining variables for multi-responses from the data set. These are described in Chapter VII of this manual in connection with other commands for multi-responses.

If you are using a ***GESS* tabs** system data set then the variable names, the values and the relevant texts are already known and obviously do not need to be re-defined. A re-definition of the VALUELABELS etc can of course be undertaken if so wished.

SINGLEQ
(also: VARIABLE)
The simplest way to build a question/variable is using the SINGLEQ statement.
Syntax:

```
SINGLEQ <varname> = [ TITLE <titletext> ] [ ALPHA ] [ [ start | * ] [
width | BINARY ] ]
[ LABELS [ AS <varname > | COPY <varname> | MAKE <number> | {
LabelEntry }*n } ]
;

LabelEntry ::=
[ <number> "Labeltext" | OVERCODE [ SUM ] [ <name> ] ValueList
"Overcodetext" ] [ LabelOption ]

ValueList ::=
<number>[ : <number> ] ... <number> [ : <number> ]
```

```
LabelOption ::=
{ [ LEVEL <number> | SORTCLASS <number> | USEFONT <font-specifier> |
.... ] }*n
```

Please see VALUELABELS for a complete list of label options.

If SUM is found directly after OVERCODE then the overcode is tallied as gross; i.e. several nuclear responses within the overcode are added together.
Example:

```
SINGLEQ Fr.23 TITLE 'Alter des Befragten' 19 LABELS
1 "bis 24 Jahre"
2 "25 bis 39 Jahre"
3 "40 bis 49 Jahre"
4 "50 and älter"
;
```

or, more complexly:

```
SINGLEQ Fr.23 TITLE 'Alter des Befragten' 19 LABELS
OVERCODE 1:2 "Junge Befragte" LEVEL 1 USEFONT "Helvetica" SIZE 14
1 "bis 24 Jahre"
2 "25 bis 39 Jahre"
OVERCODE 3:4 "Alte Befragte" LEVEL 1 USEFONT "Helvetica" SIZE 14
3 "40 bis 49 Jahre"
4 "50 and älter"
;
```

The variable (or: question) is to be called "Qu. 23", in the table it is to have the title "Alte Befragte ". It is numerically coded in Column 19 of the input data sets. The column width has been omitted; 1 is assumed.

It is numerical because the key word "ALPHA" does not follow the title.

For the codes 1, 2 and 3 label texts have been allocated for the printout. Hyphenation commands and line breaks can be contained in the label texts. The syntactical possibilities have been dealt with in detail under VALUELABELS.

OVERCODEs are a comfortable method of grouping together individual codes. They replace the additional effort of transforming the variable into a multi-response and forming new groups using RECODE or IF statements. In the second version two OVERCODES are generated in the label part.

After the key word OVERCODE there can be an overcode name which can be used to refer to it; see below SORTCLASS. The codes which are to belong to the OVERCODE are defined directly afterwards which can be done using the values (e.g. 1 : 19) or individual values in any order or a mixture of both.
Example:

```
OVERCODE O001 4 2 3 7 9:11 22 "Erster Overcode" USEFONT ...
```

Overcodes can be suppressed in TABLE = ... tables.
For this case:

```
NOOCINHEADER = [ YES | NO ]
```

and

```
NOOCINSTUB = [ YES | NO ];
```

Using the key words AS or COPY labels which have already been defined can be referred to.
Example:

```
SINGLEQ Fr.29 TITLE "Alter der Ehefrau" 24 LABELS AS Fr.23;
```

The key word AS causes in this case the variable to have the same labels as its predecessor Qu. 23. The variable which is referred to by the AS clause must exist and have labels. Parallel to this a label copy can also be requested: LABELS COPY. See also the difference to using USELABELS or COPYLABELS.

Using the key word MAKE a list of labels (1 bis n) can be produced (e.g. for input purposes).

A short note on the syntactically permissible key word BINARY instead of the column width: this can only be used in connection with a COLBININFILE. Then the 16 bit of the COLBIN column is interpreted as unsigned binary numbers whose value is ascertained and then allocated to the variables.

Instead of the explicit naming of the initial column an asterisk is permitted: then the variable is read directly after the last variable to have been defined. This enables the easily movable naming of whole variable blocks.

OPEN

The ***GESS*** system can also process and code the responses to open questions by designating the variables in the SINGLEQ to be OPEN. ***GESS*** questionnaire and input programmes then open a text window for the input of open responses.

Syntax:

```
SINGLEQ <varname> = [ TITLE "Titelstring" ] OPEN;
```

or

```
VARIABLE <varname> = [ TITLE "Titelstring" ] OPEN;
```

The responses to open questions are written in a separate output file (see OPENQFILE). These files are simple ASCII- or ANSI text files which initially contain the verbatim response texts. After coding (e.g. using ***GESS*** **OpenCode)** they also contain the codes which have been allocated to the relevant texts. For this there is the "classical" ***GESS*** **OpenFormat** and a newer NEWOPENQFORMAT or NEWOPENFORMAT = YES;), the latter containing the information as a CSV file delimited with tabulators.

Each row in a "classical" OPENQFILE contains a position "Codes ( ) ". The codes can be added between the parentheses.

During processing by ***GESS*** **tabs** all the OPENQFILEs are read before the DATAFILEs are read and processed, and the allocated codes are stored in an internal dictionary. During reading of the data from the DATAFILEs the OPEN variables are filled with the values from the dictionary.

For table printouts the OPEN variables can also be given VALUELABELs. For further processing see also COMPUTE LOAD.

Special Variable Types, Multi-Responses

Multi-responses are stored in ***GESS* tabs** special variable types which are either formed directly from the input (MULTIQ, DICHOQ) or can be built by the user from elementary variables (VARFAMILY, VARGROUP). Over and above this ***GESS* tabs** also generates "crossed" variables from several elementary ones, in which for example the single combination of sex and age are contained. For the command for this see CROSSVAR. The compilation of different target groups for stubs or banners is achieved most easily using the GROUPS command.

All multi-response variables can be used for the formation of tables as elementary variables. They can even flow into means or measured values and they can be recoded using RECODE or GROUPRECODE, but they cannot be used to calculate; calculations can however be made using their components, the elementary variables.

The concept "Missing Value" is always imprecise in connection with multi-response variables. It is often at its clearest if the area of application is explicitly allocated with a filter statement (see FILTER in the Chapter: MISSING VALUES). The alternative is the use of AUTONOANSWER.

VARFAMILY

A family is a group of variables with a shared amount of characteristics, e.g. the first, second and third response to a question. These variables can be made into a VARFAMILY which is evaluated instead of the individual variables.

Example:

```
VARFAMILY item = item1 TO item4;
TABLE = item BY alter;
```

The VARFAMILY automatically has the same VALUELABELs as the first variable used in it. The VARFAMILY can however also have its own VALUELABELs. Using VARFAMILYs there are no arithmetical operations or RECODES permitted. A VARFAMILY can be seen in regards to content as delivering several values one after the other to the evaluation process.

VARFAMILYs can also appear as arguments of a VARFAMILY statement. If for example it should be the case that in the VARFAMILYs (or MULTIQs, see below) GOOD and BAD have the relevant positive and negative responses to a product, then with:

```
VARFAMILY GoodAndBad = GOOD BAD;
USELABELS = Good;
```

a VARFAMILY can be produced that contains both the positive and the negative responses.

MULTIQ

(**also FAMILYVAR**)

Alternatively the variable family can also be generated directly from the input. This makes the individual variables invisible to the user:

Syntax:

```
MULTIQ <varname> = [NOINPUT ] [ TITLE <titlestring> ] [ ALPHA ] [
start | * ] len [width]
[ LABELS { AS <varname> | { value <text> }*n } ] ];
```

Example:

```
MULTIQ WichtPolit = TITLE "Wichtige Politiker in Hamburg"
20 10 2
LABELS
1 "Voscherau"
2 "Perschau"
....
;
```

As an expansion on the syntax described above the OVERCODEs can also be defined in the LABELS part as well as the specifications for SORTCLASS, LEVEL and USEFONT being permitted.

In that case it is assumed that the responses to the "important politicians" are coded in the columns 20-21, 22-23, etc., up to 28-29. The values can be in double figures which is why there are two columns allowed per response.

These responses could also be punched as text and a variable family generated from five ALPHA variables.
Example:

```
MULTIQ WichtPolit = TITLE "Wichtige Politiker in Hamburg"
ALPHA
1 100 20
;
```

In this case the names of politicians are punched in the fields 1-20, 21-40, etc. which makes coding by hand superfluous. If using input from a COLBIN file then the key word ALPHA can obviously not be used.

Generally the use of an asterisk instead of the initial column is processed the same as in a SINGLEQ. MULTIQs can also be defined as relocatable.

Internally FAMILYVARS are a list of nuclear variables which can be referred to simultaneously. The labels, texts and VARTITLE of the FAMILYVAR are inherited on to the nuclear variables below it. This plays a role in the output in SPSS syntax as for SPSS the individual variables are exported.

NOINHERITTEXT
NOINHERITTITLE
Syntax:

```
NOINHERITTEXT  = [ YES | NO ];
NOINHERITTITLE = [ YES | NO ];
```

In March 2007 at the client request I built in an inheritance of VARTEXT and VARTITLE for VARGROUPS and VARFAMILIES. This was very practical for output as SPSS syntax but it turned out to be a hindrance in other cases. There is thus now a new switch to turn this back off.

MAKEFAMILY
Syntax:

```
MAKEFAMILY <name> = <value>;
```

Using MAKEFAMILY a family is generated with value response possibilities.

EVALFAMVALONCE
Syntax:

```
EVALFAMVALONCE <Varlist> = YES or NO;
```

(EvalFamValOnce = EVALuate FAMilyvariables VALues ONCE). Using VARFAMILYs it can make sense according to the content dimension of the variable family to evaluate the variables with several identical codes only once in the family or all responses regardless of the identity of responses already made. The parameter YES means that all the values of the variable family per case are evaluated only once; this conforms to the preset. The parameter NO causes values which are repeatedly identical to be evaluated per case. The VARLIST can only contain names which are also in the VARFAMILYs.

IGNOREMULTIQOVERFLOW

Syntax:

```
IGNOREMULTIQOVERFLOW = [ YES | NO ];
```

Variables of the type MULTIQ or FAMILYVAR have a maximum number of values which they can take on board. If more values are to be stored in a MULTIQ than the maximum number allowed the program run as of Version 3.0 will be aborted with a RuntimeError; in earlier versions the remaining values were silently ignored. Using the switch YES the old behaviour can be restored.

INDEXVAR

Syntax:

```
INDEXVAR <name> = <varlist> BY <variable>;
```

Using the construct INDEXVAR it is possible to build ARRAYs of variables which can be referred to using an associated nuclear variable.

If in a survey on three makes of car there are three rotations and Opel was first in the first rotation, third in the second and second in the third then an INDEXVAR could be used directly for tabulation or de-rotated variables could be simply generated using COMPUTE LOAD.

Example:

```
INDEXVAR OpelIndex = Gesamt.1 Gesamt.3 Gesamt.2 BY Rotation;
...
COMPUTE LOAD OpelGesamt = OpelIndex;
```

or also:

```
TABLE = #KOPF BY OpelIndex;
```

INVINDEXVAR

Syntax:

```
INVINDEXVAR <name> = <varlist> BY <variable>;
```

Is exactly what it says on the tin: the inverse of INDEXVAR! It delivers the index of a sought code in a list of variables (index position, based on 1)

CROSSVAR

Using CROSSVAR special variable families can be produced which contain all the characteristic combinations of all the variables involved. This can be used to present multiple cross tables in TABLE for example. If one were to define:

```
CROSSVAR "Alter * Geschlecht" = Alter Geschlecht;
```

then

```
TABLE = Parteipräferenz BY "alter * Geschlecht";
```

can directly follow to present the dependence of party preference on the combination of age and sex. The VALUELABELS of the variables used to build the CROSSVAR are carried over respectively. CROSSVAR can also take in several variables.

COMBINEDVAR

COMBINEDVAR produces a VARFAMILY which contains all the individual characteristics of the individual variables next to each other.

```
COMBINEDVAR X = Alter Geschlecht;
```

produces for example a variable family with which a table can evaluate age and sex simultaneously next to one another. COMBINEDVAR is also suitable for allocating one variable to another including its VALUELABELS.

VARGROUP

defines a group of variables which are to be evaluated together. Usually VARGROUP is used to group individual variables which build a 0/1 group of multi-responses together.

Example:

```
VARGROUP Items = ( item.1 item.2 item.3 item.4 ) EQ 1;
```

In front of the equals sign there is the name of the variable group. Variable group names must conform to syntactic rules for variable names, in particular there cannot be a variable with the same name as the variable group. After the equals sign there is the list of the simple (nuclear) variables in parenthesis which are to be formed into a group. The number after "EQ" states which characteristic in the list of the original variable is to be taken as the "response" in the sense of the variable group. There are no arithmetical operations which are permitted as arguments in the variable families. RECODE or RANGES are possible but are used as operations for all single nuclear variables. Variable groups can otherwise be treated as variables and for example can have VARTITLE or VALUELABELS.

DICHOQ

also: GROUPVAR

Variable groups can also be generated directly from the input without making the individual variables visible.

Syntax:

```
DICHOQ <varname> =
[ NOINPUT ]
[ TITLE <titlestring> ]
{ start [ width ]  <labeltext> [ SORTCLASS <number> ] [ LEVEL
<number> }*n
EQ value;
```

Example:

```
DICHOQ WichtThem =
TITLE "Wichtige Themen des Wahlkampfs"
22 "Gesandheitspolitik"
23 "Asylpolitik"
24 "Schulpolitik"
25 "Wirtschaft"
EQ 1;
```

NB: Syntactically a field width can follow the start position. This means however that a DICHOQ can also not have a label which can be interpreted numerically as the parser then does not see it as a label text but rather as a field width!! (Quotation marks do not help here as a number can also be put into quotation marks!).

Generally the use of an asterisk * instead of the start column means the same as in a SINGLEQ. DICHOQs can also be relocatabley defined. The example above could also be written as follows (as long as the last variable ends in column 21):

Example:

```
DICHOQ WichtThem =
TITLE "Wichtige Themen des Wahlkampfs"
*  "Gesundheitspolitik"
*  "Asylpolitik"
*  "Schulpolitik"
*  "Wirtschaft"
EQ 1;
```

It is also permissible to include OVERCODE, font information, LEVEL or SORTCLASS.

Example:

```
DICHOQ WichtThem =
```

```
TITLE "Wichtige Themen des Wahlkampfs"
OVERCODE 1 2 "Gesundheits- and Asylpolitik" LEVEL 1 USEFONT
"Helvetica-Bold" size 11
*  "Gesundheitspolitik"
*  "Asylpolitik"
*  "Schulpolitik"
*  "Wirtschaft"
EQ 1;
```

The codes from DICHOQ always count as of 1, increment of 1.

A different syntax is valid for input from a COLBIN file:

```
DICHOQ <varname> = [ TITLE <titlestring> ] { column:value  <label-
text> }*n ;
```

Value can take on the value from 0 .. 9, X and Y. Here the bits in a COLBIN column are described.
Example:

```
DICHOQ WichtThem =
TITLE "Wichtige Themen des Wahlkampfs"
22:0 "Gesundheitspolitik"
22:1 "Asylpolitik"
22:X "Schulpolitik"
22:Y "Wirtschaft";
```

The use of asterisks for following columns has no use in COLBIN. In COLBIN the concept of the following column is somewhat imprecise as it can refer to the next bit.

MAKEGROUP

Syntax:

```
MAKEGROUP <name> = <value>;
```

Using MAKEGROUP a group variable is generated with n individual variables.

SPSSGROUP

Syntax:

```
SPSSGROUP <name> = <familyvarname>;
```

Using SPSSGROUP a group variable is generated which contains enough variables to represent all values between 1 and the maximum label value. (Maximum of 300). Then the content of the MULTIQ "familyvarname" is carried over to the newly generated DICHOQ.

SPSSGROUPLABELS
SPSSGROUPLABEL1
SPSSGROUPLABEL0

A SPSSGROUP comprises a row of nuclear variables where the Code 0 or 1 shows whether the relevant value is "set". The SPSSGROUP statement has now (as of Version 4.0.2) been expanded so that these nuclear variables can be allocated information from the label of the relevant code of the source variable (MULTIQ).

Syntax:

```
SPSSGROUPLABELS = [ YES | NO ];
SPSSGROUPLABEL1 = <TEXT>;
SPSSGROUPLABEL0 = <TEXT>;
```

SPSSGROUPLABELS switches this feature on or off and the texts for the characteristics 0 and 1 can be set using SPSSGROUPLABEL1 and SPSSGROUPLABEL0.

a) if SPSSGROUPLABEL1 <> '' then the text of the relevant labels of the source MULTIQ is sunk in the VARTITLE of the nuclear variable of the SPSSGROUP and Code 1 gets the value of the SPSSGROUPLABEL1.
if SPSSGROUPLABEL0 <> '', then this will become the text of the label for Code 0.

b) if SPSSGROUPLABEL1 = '', then the text of the relevant label of the source MULTIQ becomes the text of the label for the characteristic 1.

Clear as mud?

CONDENSESPSSGROUP

Syntax:

```
CONDENSESPSSGROUP = [ YES | NO ];
```

If this switch is set to YES then the codes of the MULTIQ which are to be transformed in a SPSSGROUP are set to the value 1 - number of labels recoded. In effect this is also the way to reduce a MULTIQ with a systematic codeplan which contains many empty spaces to a dichotomous variable, which contains exactly the amount of nuclear variables (columns) needed to build all the labels. The labels in the SPSSGROUP comprise texts from the labels of the MULTIQ which prefix the relevant codes from the MULTIQ.

GROUPCOUNTS

Syntax:

```
GROUPCOUNTS <Varlist> = [ YES | NO ];
```

Preset: YES

If VARGROUPS are built there can always be cases where none of the individual variables are valid. If a table is evaluated per case it depends on the formulation of the question as to whether such a case is to be counted or not. If tables are calculated on the basis of responses this has no relevance.

Using GROUPCOUNTS = YES such a case is counted as valid in regard to the VARGROUP and flows into the basis of the table. Using GROUPCOUNTS = NO such a case is counted as invalid and may not flow into the basis of the table. Such a case would obviously never flow into the individual cells of a table (as it cannot be subordinated). See also FILTER.

INIT

Syntax:

```
INIT <varlist> = <value list>;
```

Using INIT variables can be given a statistical initialisation. This is particularly interesting for multi-response variables. A VARFAMILY is considered to be a list of its nuclear variables here.

NB: INIT is static and is only used during the compilation of a program, i.e. the test variable (test in the example) must not be altered at a different point in the script: then it no longer works.

GROUPS

If the individual (nuclear) variables from which the variable groups are to be formed are not yet present then the GROUPS command is often the more practical alternative as the naming and the more complex rules for forming groups can be formulated more clearly in the GROUPS command. If a certain target group is to stand out in a series of tables then a collection of target groups is described as follows:

```
GROUPS Zielgruppen =

| "jüngere wohl-haben-de Männer" :
Alter le 45 und einkommen ge 3000 und geschlecht eq 1

| "ältere wohl-haben-de Männer" : USEFONT "Courier" size 15
Alter gt 45 und einkommen ge 5000 und Geschlecht eq 1

| "jüngere wohl-haben-de Frauen" :
Alter le 45 und einkommen ge 3000 und geschlecht eq 2

| "ältere wohl-haben-de Frauen" :
Alter gt 45 und einkommen ge 5000 und geschlecht eq 2
;
```

These GROUPS statements produce a VARGROUP with the name "Zielgruppen" which has the four characteristics described. After each colon there can be any complex logical stipulation as to the membership of a case. The parts of the stipulation can be combined with AND and OR, parenthesis can be used. The logical stipulation can contain text constants.
Syntax:

```
GROUPS <Varname> =
{ | "Labeltext ..." [ LEVEL <number> ] [ USEFONT <Fontname> [ SIZE
<number> ] ] : <log. Bedingung>  }*n ;
```

The usual rules concerning indication and splitting are valid for the label texts, see also VALUELABELS. Individual labels can be given a label font (see above).

ASALPHA

Syntax:

```
ASALPHA <varlist>  = [ YES | NO ];
```

Using the characteristic ASALPHA ***GESS* tabs** can be told to treat OPEN variables as ALPHA variables during processing for output; i.e. the texts from the OPENQFILE become LABELS for the OPEN variables. This can save coding with small amounts.
Example:

```
ASALPHA var1 var27 f23 = yes;
```

VARIABLES

Using the VARIABLES statement a series of variables can be generated which are stored together in the data set:
Syntax:

```
VARIABLES <varname><varnumberstart> TO <varname><varnumberend> = [
start | * ] [width];
```

Example:

```
VARIABLES Item1 TO Item13 = 33 2;
(Item1 ist in Sp. 33-34, Item2 in Sp.35-36, etc. )
```

The asterisk in place of the numerical start position states that the block of variables is to be positioned directly after the variable which has been defined last.

COLUMNVARS

This is an alternative method of building a series of variables. The initial column of the variable becomes a component part of the name.
Syntax:

```
COLUMNVARS <nameprefix> = start - end [width];
```

Example:

```
CARD = 1;
COLUMNVARS S1. = 1 - 80;
CARD = 2;
COLUMNVARS S2. = 1 - 80;
```

Here 160 column variables are created; 80 on card 1 and 80 on card 2. The variables on card 1 are called S1.1 - S1.80, and those on card 2 are called S2.1 - S2.80.

CARDS

CARDS discloses how many records or rows constitute the case.

Example:

```
CARDS = 2;
```

Preset is CARDS=1, i.e. for data sets which comprise one row the specification is not necessary.

CARD

discloses in which row or "card" the then following variables or weight is to be found. Preset is on CARD=1.

CARD and CARDS refer to the DATAFILE (and thus automatically the COPYFILE). Relevant commands are also available for ASCIIOUTFILE, COLBININFILE and COLBINOUTFILE.

ASCIIOUTCARDS
ASCIIOUTCARD

The number of cards and the preset of the present card for the output of variables in ASCII format in the ASCIIOUTFILE.

COLBININCARDS
COLBININCARD

Definition for the data set to be read in COLBIN format.

COLBINOUTCARDS
COLBINOUTCARD

Definition for the data set to be written in COLBIN format.

VARNAME

defines a variable and its position in the DATAFILE if necessary in the COPYFILE or also in the COLBININFILE. If a variable in a particular "row" is to be referred to then it is preceded by the key word CARD or COLBININCARD (see below).

Example:

```
VARNAME = Alter 101 1;
```

Age is coded in column 101, length = 1.

Example:

```
CARD = 3;
VARNAME = ITEM37 44 2;
```

ITEM37 is coded in column 44-45 of card 3. Here an asterisk can replace the start column and then the variable is positioned directly after the variable produced last.

A blank space can also be used as part of the variable name but then the variable must be placed in quotation marks.
Example:

```
VARNAME = "Biologisches Alter";
```

If the VARNAME statement refers to variables in a COLBIN file then the columns must not contain multi-punches and only punches from '0' - '9' are permissible. Columns that do not fit these requirements lead to input errors. In these cases the column number is omitted and the input is defined using COLBININ.

The input files in the ASCII file may contain REAL or INTEGER values. REAL figures must explicitly contain the decimal point. Input data which does not contain valid figures must use VARNAME statements in the ALPHA variation (see below). Variables in COLBIN files may not extend over several columns; as there is however no equivalent in COLBIN format for the decimal point only INTEGER values can be processed.

If a variable which does not yet exist is to be generated and output in the COPYFILE then a variable position must be provided which may not yet be present in the input data set. The relevant key word is NOINPUT. If e.g.: the input data set is 70 columns long and a new variable NEWVAR is to be positioned in columns 80-82 then the following statement must be written before the COMPUTE statement:

```
VARNAME = NewVar 80 3 NOINPUT;
```

The program would fill the area 71 - 79 with blanks, the new value would be positioned in columns 80-82 and the rows of the OutFiles would be 82 characters long. This somewhat strange procedure allows data sets to be extended and altered without writing explicit ASCIIOUT commands.

```
VARNAME ALPHA =
```

Using the key word ALPHA in front of the indicating symbol a variable can be defined as alphanumeric during input. Then the input field in the data set is not interpreted as numeric but as a character string which can be of any length; all characters in the character string are significant.

Nevertheless the variable contains a numerical value which reflects the order of appearance of varying character strings. The first character string to appear is represented at 1, the second at 2 etc.

The mechanism of the ALPHA variables can be described as follows: using ALPHA variables the program expects the VALUELABEL text in the DATAFILE and not the variable value. If a text is found which does not conform to a known label a new label is generated with its own variable value. If a variable does not yet have a label then the new value begins with 1. Capital letters are irrelevant for the identity of the labels. For output the spelling is used which was found at the first appearance. Labels defined with VALUELABELS are considered to be the first appearance.

It is also syntactically possible to process these variables with RECODE or COMPUTE; this occurs solely at the user's own risk!

As far as the character strings contain the usual dividers in the input file the dividing rules from ***GESS* tabs** will be used during output.
Example:

```
VARNAME ALPHA = Interviewername 1 32;
```

Predefined variables:

SystemFileNo	contains the number of the file being processed. This allows the splitting of the total evaluation according to the source file if several files are to be read
SystemWeight	contains the valid weight for the case
SystemCaseNo	contains the case number starting with case number 1.
SYSMISS	a variable which always has the value Missing

SystemIntervNo	contains in CATI etc. the Interviewer Number
SystemDialState	also only for CATI
SystemStudioNo	also only for CATI
SystemLanguage	also only for CATI
SystemAppoint	also only for CATI
SystemAbandon	also only for CATI
NIL	empty variable necessary for e.g. TABLE ADD

The attempt to generate variables with these names leads to an error.

VARTITLE

Syntax:

```
VARTITLE <VarList> = "String";
```

Every variable can be allocated another title for the table headline. The command refers to the variable directly preceding or explicitly named.

Example:

```
VARTITLE = "Wahlabsicht bei der kommenden Landtagswahl";
```

The string may contain line feeds. VARTITLE is subject to the same dividing rules as described below in VALUELABELS.

ADDNAMETOVARTITLE:

Syntax

```
ADDNAMETOVARTITLE = [ yes | no ];
```

Due to special client demand:
When using YES the variable name is printed as well the VARTITLE in cross tables.

COPYTITLE

Syntax:

```
COPYTITLE <varlist> = <variable>;
```

All variables in <varlist> (in some cases the last defined variable) contain a reference to the VARTITLE of <variable>.

VALUELABELS

Syntax:

```
VALUELABELS <VarList> = [ ADD ]
{ LabelEntry }*n ;

LabelEntry ::=
[<number> "String" | OVERCODE [ SUM ] [<name>] { <number> [ :<number>
] }*n "String" ] [ LabelOption ]

LabelOption ::=
{
[ SORTCLASS <number> ] | [ LEVEL <number> ] | [ USEFONT
<fontspecifier> ] | BOTTOM
| SINGLE | SLICE | ALWAYS | NOCITATION | NEWPAGE | NEVER
| [ CELLELEMENTS ( <cellelements> )] | [ RECODE <value> ] ]
}*n
```

ADD	if this option is set then the existent labels are not changed but remain. A LABELS statement always produces only one value label object which is then linked to all the variables in the <varlist>. If LABELS ... ADD is used it is the same procedure; here the internal logic leads to all variables in the <varlist> getting the labels of the first variables in the list expanded/altered by the additional components of the statement. This however only makes sense if the LABELS <varlist> = ADD ... statement refers to a list of variables which already has identical labels. In other cases this leads to unexpected results. The LABELS ... ADD function has therefore been restricted to variable lists with the length of 1. More than one variable at this point causes the syntax error 528.
SORTCLASS	Sort class. Preset to 0, range of values -255 - +255. The sort command causes the labels to be sorted first according to the SORTCLASS and then within the same SORTCLASS according to the sort criterion (e.g. MEAN). Note regarding AUTOOVERSORT: if an OVERCODE has a SORTCLASS between 1 and 100 then the SORTCLASS also affects the elements in the overcode (codes and hierarchically subordinated overcodes). SORTCLASS values above 100 only affect the overcode. If e.g. an OVERCODE SUM across all responses is to be sorted at the end of a table then a SORTCLASS > 100 has to be defined. Using SORTCLASS 99 for example all the responses would otherwise be presented after the overcode according to the definition of AUTOOVERSORT.
LEVEL	counting level usually 0, i.e. in MULTITOTAL the following are valid: 1 if not to be evaluated in MULTITOTAL also controls the output in TABLE tables 99 test against ROWMINIMUM/COLMINIMUM not applicable 98 not to be printed if row/column frequency = 0 252: never print label, always suppress
NOCITATION	never use this label with the @-option to quote in question texts.
USEFONT	Font information
CELLELEMENTS	command for differing cell elements for the row after this label
SLICE	see SLICE in the TABLE statement: divide in the edge zone of the table before this label
NEWPAGE	ALWAYS print this label on a new page (alias: PAGE)
BOTTOM	for simple codes: always sort at the end of a SORTCLASS ("other") for CATI/CAPI: always sort this label at the end of all response possibilities, also for RANDOM/ROTATE
SINGLE	only CATI/CAPI/data entry: this category must not appear with any other category. e.g. "none of the above".
ALWAYS	only CATI/CAPI/data entry: these labels are always permitted regardless of RESTRICT commands
RECODE	alternative to RECODE commands recodes can also be undertaken in label lists. This has two prerequisites: ***GESS* tabs** (not REPORT or CATI), and LABELRECODE is previously set to YES in the script. These RECODES are inherited on to LABELS COPY or LABELS AS.
NEVER	Never show these labels during the survey.

VALUELABELS is used to define labels for individual codes. The definition of the VALUELABELS refers either to the last variable defined or an explicitly named variable. In SINGLEQ, MULTIQ or DICHOQ all components are basically valid.
Example:

```
VALUELABELS alter1 alter2 =
1 "unter 18 Jahren"
2 "18 bis 24 Jahre"
3 "25 bis 39 Jahre"
4 "40 bis 64 Jahre"
5 "65 Jahre und älter"
;
```

The string of commands is concluded with a semi colon. The quotation marks are only necessary if the individual label texts contain special characters, otherwise following form is sufficient:

```
1 männlich
2 weiblich
```

etc.

VALUELABELs can be written in any order. For the printing of TABLE cross tables the order is the standard order for printout. Thus if the category "female" is to appear before the category "male" in the table then the following can be written:

```
VALUELABELS =
2 weiblich
1 männlich
;
```

If a variable in a table also appears in the head column then it can be necessary to divide individual words in the VALUELABELs. For this purpose a minus sign or a hash (#) can be added as both signs are recognised as dividers.
Example:

```
1 "männ-lich"
2 "weib-lich";
```

If dividers, blanks or special characters are within texts then the VALUELABELS must be placed within quotation marks. The special characters which serve as dividers can be chosen at will. Preset are:

```
SPLITCHAR = ,-,;
SPLITCHARSTAY = ,#';
```

For most users the following have become practical standards:

```
SPLITCHAR = ,~,;
SPLITCHARSTAY = ,-';
```

The old preset is often somewhat impractical ('-' and '#') and is thus therefore almost always replaced by '~' and '-'. If AFM files are used and thus LATIN1 is automatically valid as script encoding then this is automatically preset.

If printing does not require division then the SPLITCHAR are removed beforehand. Further dividers are recognised: blank and comma. These symbols are however not removed. The SPLITCHARSTAY serves as an "irremovable" divider; i.e. even if no division is necessary the divider is printed.
Example:

```
VALUELABELS = 1 "Auto-mobil#Zeit-schrift" ;
```

will be printed as Automobil-Zeitschrift as the value label of a page group.

SPLITENTRIES

```
SPLITENTRIES = <filename>;
```

It is very easy to produce a "dividing" dictionary. If a list is constructed like so

```
Nie~der~sachsen
Bundes~land
Wahl~ab~sicht
Weiterfüh~ren~de
Polytech~ni~sche
Hoch~schul~reife
Selbst~ständige
Aus~zu~bil~den~de
wahr~schein~lich
```

```
Eigentums~woh~nung
Mecklen~burg
Württem~berg
Branden~burg
Haupt~schule
Ange~stellte
Vor~pommern
```

and this is stored in a text file which is called up using SPLITENTRIES then the words in the file will be divided in the table as they have been stored in the dictionary. This can be particularly useful in projects containing many label texts from SPSS-SAV-files. The SPLITCHAR defined is processed.

The back slash "\" is used in a LABEL row to indent particularly interesting characteristics of a table for emphasis:

```
VALUELABELS =
1 "erste Kategorie"
2 "zweite Kategorie"
3 "\dritte, wichtige Kategorie\"
4 "vierte Kategorie"
5 "fünfte Kategorie";
```

The back slashes at the beginning and the end of the row introduce empty rows before and after the row.

Individual characteristics can be allocated to a font for printout and in the above example the third category could also be made to stand out even more (in this case with Palatino in bold and particularly large):

```
VALUELABELS =
1 "erste Kategorie"
2 "zweite Kategorie"
3 "\dritte, wichtige Kategorie\" USEFONT "Palatino-Bold" SIZE 17
4 "vierte Kategorie"
5 "fünfte Kategorie";
```

The font has to be valid for the printer to be used (see also the explanation for USEFONT). The font is used to print the label text and also all the cell contents in the relevant row or column. An inherited label font is also only valid subordinate to any fonts for the contents of the cell. The inheritance of the font is switched on using INHERITFONT. (compare with. USEFONT, INHERITFONT)

The variable characteristics can be allocated to different SORTCLASSEs for printing. Sorting is always carried out within the SORTCLASS and usually all labels have the SORTCLASS 0. Possible SORTCLASSEs are from -127 - 127. A standard use is to sort the characteristics of a variable in descending order of frequency whilst always leaving the category "other" at the end.
Example:

```
VALUELABELS y =
1 'Marke X'
2 'Marke Y'
3 'Marke Z'
4 'Andere Marken' SORTCLASS 1 LEVEL 1
CELLELEMENTS( ABSOLUTE COLUMNPERCENT )
;
....
CELLELEMENTS = COLUMNPERCENT;
TABLE = Kopf BY y SORT ABSOLUTE DESCEND;
```

In connection with MULTITOTAL it is not necessary to stipulate an evaluation level. Normally all characteristics have LEVEL 0. If the LEVEL is set to <> 0 the relevant characteristics will be ignored when tallying the total. Level values: 0 … 127.

In the VALUELABELs statement OVERCODE can be used just as in the LABELS part of SINGLEQ or VARFAMILY statements.

LABELRECODE

Syntax:

```
LABELRECODE = [ YES | NO ];
```

LABELRECODE controls whether RECODE is permitted as a label option in the script. LABELRECODE is only permitted in ***GESS* tabs** and not in CATI or REPORT.

COPYLABELS
USELABELS

Using COPYLABELS and USELABELS variables can be allocated the VALUELABELS of other variables.

Syntax:

```
COPYLABELS <Varlist> = Variable;
```

or

```
USELABELS <Varlist> = Variable;
```

Example:

```
COPYLABELS a b c = MarkeX;
```

COPYLABELS makes a real copy. USELABELS stipulates the multiple use of label texts which have only been stored once, i.e. these variables use the same copy of the label text structure. This saves on storage space.

During tabulation the label texts can be used for different types of variables using USELABELS; i.e. simultaneously for simple variables and multi-response variables. When being used in questionnaire programs (e.g. CATI) or for data entry (e.g. Input) it is not only the label amount which is used repeatedly but also the monitor object which conducts the data entry. As the monitor object for multi-response variables has to behave differently to that of for single variables the multiple use of labels using LABELS AS or USELABELS is only permitted for variables of the same type.

So for example:

```
VARIABLE fx = * LABELS
1 "eins"
2 "zwei"
;
FAMILYVAR ff = * 5 LABELS AS fx;
```

is permitted for tabulation script but a syntax error occurs during compilation by means of a questionnaire or data entry program.

LABELFORMAT

Syntax:

```
LABELFORMAT <varlist> = <string>;
```

Usually labels are allocated to the variable codes. ***GESS* tabs** can however also produce labels directly from the figure values whilst printing a table. This production is caused by the LABELFORMAT statement.

Assuming that a date in the form of DDMMJJJJ is stored in a variable in numeric form then this value can be made legible by a LABELFORMAT. The figure value e.g. 26062005 is replaced by:

```
LABELFORMAT Datum = "##.##.####";
```

and printed as

```
26.06.2005
```

Usually a preceding zero is also generated i.e. 5072005 becomes:

```
05.07.2005
```

If this is not required then the TABLEFORMAT NOZEROFILLINLABEL can negate this.

GENERATELABELS

Syntax:

```
GENERATELABELS <varlist>;
```

In LABELFORMAT strings the @varname constructions are also evaluated. If the LABELFORMAT contains an at-symbol and then a valid variable name then this character string is replaced by the text of the label which conforms to the valid value of the variable. This cannot occur at the time of printing the tables as at that point none of the variables has a valid value.

Thus if the @varname variant is used in the LABELFORMAT then the relevant variable(s) must also be packed into a GENERATELABELS command.

SORTCLASS

Usually the SORTCLASS information is given to labels and overcodes in the VALUELABELS statement or the LABELS part of the SINGLEQ, DICHOQ or MULTIQ statement. There are however cases where it makes sense to provide the SORTCLASS information later in the text. This is particularly the case where a common LABEL LIST is used for several variables but the LABELS and OVERCODEs are to be sorted differently.

Syntax:

```
SORTCLASS <varname> OVERCODE <name> = <number>;
```

Here the OVERCODE is allocated the SORTCLASS <number> and all labels belonging to the OVERCODE receive the SORTCLASS <number> + 1. In this way OVERCODEs and the relevant label positions can be sorted in a table. Prerequisite is the agreement of a name for the OVERCODEs when they are generated so that they can be referred to in the SORTCLASS statement.

Individual labels can also be given a new SORTCLASS:

Syntax:

```
SORTCLASS <varname> LABELS <number> [ <number> ... ] = <number> ;
```

ZONEINPUT

Syntax:

```
ZONEINPUT <varname> = [ MEAN | SUM | COUNT | MIN | MAX ]
<start> <zonewidth> <end>
<varoffset> <varwidth>
{ SELECT <offset> <string> } *n ;
```

Special input commands for ASCII data. If there is data with information blocks of the same structure which repeatedly occur then the information can be extracted most efficiently in this way.

The extractable information is MEAN, SUM, COUNT, MIN or MAX. As an option any number of SELECT clauses can be tested as to whether the relevant zone in the analysis should be included; more than one SELECT clause is then linked internally using AND.

Example:

```
VARIABLE ZoneMax =;
ZONEINPUT ZoneMax = MAX
11 10 50 // Zonen von 11 bis 50, zonenbreite = 10
0 2      // auszuwertender Inhalt in 11..12, 21..22, 31..32, 41..42
```

```
SELECT 2 '123';  // werte nur die Zonen aus mit '1', '2' or '3' in
sp. 13, 23, 33 and 43
```

COLBININ

Syntax:

```
COLBININ <varname> = { | value < column : code }*n };
```

Using the COLBININ statement individual characteristics of an existing variable are "bound" to certain bits in one or more COLBIN columns. The following sequence generates the variable age and sets the characteristic dependent on certain bits in two COLBIN columns.

Example:

```
VARNAME = Alter;
COLBININ Alter =
| 1 < 22:9
| 2 < 22:X
| 3 < 22:Y
| 4 < 23:0
| 5 < 23:1
| 6 < 23:3;
```

If several of the bits 22:9 - 23:3 are set then the variable has the characteristic found last i.e. 23:1 predominates over 23:0 or 22:Y.

COLBINOUT

The counterpart of COLBININ is COLBINOUT. In the first form it is very similar to COLBININ described above:

Example:

```
COLBINOUT Alter =
| 1 > 22:9
| 2 > 22:X
| 3 > 22:Y
| 4 > 23:0
| 5 > 23:1
| 6 > 23:3;
```

The COLBINOUT statement is also used to build VARFAMILYs and VARGROUPs on COLBIN multi punches as the variables can be multi-response variables. The allocation of the characteristics in variable families is trivial; in VARGROUPs the individual variables are virtually built on the characteristics 1, 2, ... n.

There is an additional form of the COLBINOUT statement:

Example:

```
COLBINOUT Alter = 16;
```

The characteristics 0 ... 9 of the variable age are placed in column 16, for all the other values column 16 remains empty. A whole series of variables can also be placed in successive COLBIN columns:

Example:

```
COLBINOUT Item1 TO Item20 = 41;
```

Here the variables Item1 - Item20 are consigned to columns 41 - 60. Whole positive figures can be placed in COLBIN fields over several columns:

```
COLBINOUT Item1 TO Item10 = 11 2;
```

The item variables are placed in the columns 11 and 12, 13 and 14, etc. up to 29 and 30 n as long as their values are whole and between 0 and 99.

For two types of coding that occur regularly there are special models for representing bits with which typical code plans can be represented more easily.
Syntax:

```
COLBINOUT <varlist> = <start> <width> BITGROUP [ 10 | 12 ];
```

BITGROUP 10 realises the following representation:

```
1.Column (<start>)     1 > 1
2 > 2
....
9 > 9

2.Column (bis len)     10 > 0
11 > 1
....
19 > 9
```

etc.

BITGROUP 12 realises the following representation:

```
1.Column (<start>)     1  > 1
2  > 2
....
9  > 9
10 > 0
11 > X
12 > Y

2.Column (bis len)     13 > 1
14 > 2
....
21 > 9
22 > 0
23 > X
24 > Y
```

etc.

ASCIIOUT

Every variable which is to appear in an ASCIIOUTFILE must be included in an ASCIIOUT statement.
Syntax:

```
ASCIIOUT <Varlist> = startcolumn [ width ];
```

or

```
ASCIIOUT ALL;
```

Example:

```
ASCIIOUT Item1 TO Item20 = 21;
```

The variables Item1 - Item20 are placed as of column 21 in the ASCIIOUTFILE. As no explicit field width has been stipulated each variable has one column, i.e. columns 21 - 40 are punched. In the <varlist> there can be simple (nuclear) or also group variables (VARGROUP/DICHOQ) or family variables (MULTIQ/VARFAMILY). These multi-response variables are treated as an explicit variable list of its nuclear basis variable for the output. OPEN variables have to already be transformed into a standard variable.

If WIDTH is not stipulated it is extracted from the ASCIIIN descriptor if the variable is read from an ASCII file. If the variable is built from for example COMPUTE then this information is missing. If the variable has labels then the highest label is taken to stipulate the column width. If this information is also missing then the default 5 is set.

Using "ASCIIOUT ALL;" every variable read from the DATAFILE is written as a copy with the same columns into the ASCIIOUT file. Columns can be read from the DATAFILE repeatedly but using ASCIIOUT ALL this can lead to chaos so there is a test for double assignment. In order to please all users I had to integrate a switch for turning this test off (see below: IGNOREASCOUTDUPL).

Analogue to the definition of variables for input an asterisk can be used instead of a numerical value for an initial column which means as much as "the next free column".

IGNOREASCOUTDUPL

Syntax:

```
IGNOREASCOUTDUPL = [ YES | NO ];
```

As described above this switches off the automatic test to avoid double column punches in ASCIIOUT ALL.

VI Data Manipulation

There are the usual commands for altering calculations, logical constructions, recoding etc. As a special feature COUNT, COMPUTE and IF also accept several resulting variables: these contain all the resulting values. Thus if for example 20 variables are to be set to 0:

```
COMPUTE item1 TO item20 = 0;
```

RECODE
allows the reprogramming of of individual variable characteristics of the variable defined last or a list of explicitly named variables.
Example:

```
RECODE 1 2 3 = 3;
```

summarises the characteristics 1,2 and 3 of the variable defined last to 3.

```
RECODE item1 item2 item3 1 = 4;
```

recodes the characteristics of item1, item2 and item3 of the variable.

also possible: `RECODE item1 TO item8 1 = 4;`

Several recodes can be summarised in one variable; individual recode groups are separated by "/". If whole numerical areas are to be reprogrammed then these areas can be defined with a colon.
Example:

```
RECODE x y z 1:5,7=1 / 6 : 10 = 2 / 11 : 15 = 3;
```

A list of RECODE commands which are divided by a / are only searched until a match is found. In this case the 7 is coded to 1 although 7 is also included in the second recode test. Commas in the VALUELIST have no meaning, i.e.; "1,5", "1 5" and "1, 5" are all interpreted identically.

If not all of the values listed are to be set to a remaining or default value then the key word ELSE can be written in the last part of the RECODE command instead of a numerical value or area.
Example:

```
RECODE x y z 1:5=2 / 6:10=1 / ELSE=99;
```

RECODE commands can also be in $-rows in input files. (compare also the explanation in Chapter XII, variable redefinition and treatment of survey waves.) Recodes which are in $-rows in the input file are only valid for the cases within this file, i.e. they lose their validity at the end of the file. Then for example several waves whose variables are in different input files and possibly also at different input positions can easily be processed if the variable characteristics are not coded identically. At the beginning of every file are the column commands (VARNAME) and the RECODEs specific to the file in order to adapt the data. Each of these rows is marked with a $ sign in column 1.

RECODEs can only be used for VARFAMILY and VARGROUP: then all the nuclear variables summarised in the VARGROUP or VARFAMILY are recoded.

If there is a variable group for example with 20 variables:

```
VARGROUP gruppe = ( item1 TO item20 ) EQ 1;
```

then

```
RECODE Gruppe 3 : 5 = 2;
```

means the same as

```
RECODE item1 TO item20  3 : 5 = 2;
```

RECODEs can be made to depend on a condition, e.g.:

```
IF ( v3 eq 44 ) OR ( 7 IN f5 ) RECODE v7 2=3 / 4=5 / ELSE = 99;
```

Instead of a constant RECODE value the variable name of a nuclear variable can also come after the equals sign.

GROUPRECODE

Using GROUPRECODE group variables can also be recoded.
Example:

```
GROUPRECODE GRR 3=5;
```

checks in the third variable of the group whether it is relevant and if yes the value of this variable is deleted and the fifth variable is set to TRUE.

Instead of the constant RECODE value after the equals sign there can also be a variable name of a nuclear variable (see above).

RANGES

Syntax:

```
RANGES <VarList> <ValueList> ;
```

Allows the summary of characteristic areas (RANGES) to single codes. This is achieved using a series of upper limits.
Example:

```
RANGES 1000 2000 3000 4000;
```

recodes all values < 1000 to 1, < 2000 to 2, < 3000 to 3, < 4000 to 4, >= 4000 to 5. The range values can also contain decimal points, thus:

```
RANGES Points 0.1 0.2 0.3 0.4 0.5 0.6;
```

As in RECODE variables can also be explicitly named. VARGROUPs or VARFAMILYs can also be used as in RECODE.

COUNT

tallies the frequency of preselected characteristics in a variable list.

Syntax:

```
COUNT <varlist> = ( <varlist> ) [ <logop> <number> | IN [ <number> :
number ] ] ;
logop ::== [ EQ, NE, LT, LE, GT, GE ]
```

The logical operators EQ, NE, LT, LE, GT, GE and IN can be used. For COUNT with IN the usual syntax of the interval is valid [<Untergrenze>:<Obergrenze>]. The number of variables are counted in which the condition "var GE untergrenze" AND "var LE Obergrenze" is valid.

If in the variable list VARGROUPS or VARFAMILYs is present then the COUNT is used on its nuclear variables.
Example:

```
VARGROUP items = ( item.1 item.2 item.3 item.4 ) EQ 1;
COUNT mittel = ( v1 v2 v3 v4 v5 ) IN [ 2 : 3 ];
COUNT n1 = ( item.1 item.2 item.3 item.4 ) EQ 1;
COUNT n2 = ( items ) EQ 1;
```

n1 and n2 are identical as the variable group items in COUNT for n2 contain the same variables as in the explicit variable list in COUNT for n1.

COMPUTE
allows the new calculation of variables by means of four basic arithmetical operations. New variables can be defined or existent variables can have their values changed.

If there is a variable in the left half of the COMPUTE statement which the compiler does not yet recognise then it is produced. This is then valid as the "current" variable. This variable is then referred to by further commands which contain no explicit variable list as e.g. VALUELABELS, RECODE, PRINTALL etc.

If a target variable in a COMPUTE statement does not exist then a variable of this name is automatically produced which in the standard case will be a nuclear variable. A further "comfortable" automation with COMPUTE LOAD etc. is that the target variable is produced in the pattern of the source if the source is only one variable, e.g. only one MULTIQ for 10 responses.

If however an OPENQ or an ASSOCVAR for example is to be COMPUTEd in a new non-existent variable then there is the problem that there is no pattern for the variable to be generated: it is for example not known how many codes are contained in an ASSOCVAR or an OPENQ. The standard case is that ***GESS* tabs** produces a nuclear variable at this point and thus values can be lost without being noticed.

At this point ***GESS* tabs** now (as of Version 4.0) generates a syntax error. It is then the up to the user to present a suitable variable.

There is however (as always) an exception: if using COMPUTE ALPHA then the first text is carried over and the further texts are ignored and the target variable automatically takes on the characteristic ALPHA.
Example:

```
COMPUTE Einkommen = einkommen1 + einkommen2;
COMPUTE Mittel = ( var1 + var2 ) / 2;
```

The dominant rule "multiplicative operations take precedence over additive operations" is valid. An alteration to the syntax parser has eliminated this annoying fault. (as of Version 4.0)

The result of the calculation should usually be stored in one variable. Several variables can be written however in front of the equals sign so that all of these variables are set to the value given.

Thus `Compute a b c = 0;` is permitted.

Special case:
If in a statement `COMPUTE len = var;` the variable `len` is an OPEN variable with the characteristic ASALPHA then the variables `len` have the length of the OPEN texts from `var`.

Functions:
Within the COMPUTE statement functions can be used for mathematical transformation. The following functions are available in the system:

LN	
EXP	
ENTIER	
ROUND	
RANDOM	
SIN	
COS	
TAN	
ABS	
DAYOFWEEK	The day of the week in a date in the form YYYYMMDD 1=Monday, 2=Tuesday etc Thus e.g. DAYOFWEEK(20061030) = 1. DAYOFWEEK(0) is today.
NEG	(negative value)

Example: `COMPUTE x = ENTIER( NEG( b / 2 ) );`

CAPI + INPUT :

`TIME ( 0 )`	system time in the form HHMMSS
`DATE ( 0 )`	system time in the form YYMMDD
`TIMER ( 0 )`	time in seconds since the program started or the last call up of `TIMER(0)`, reset the `TIMER` to zero
`TIMER ( 1 )`	time in seconds since the program started or the last call up of `TIMER(0)`, does **not** reset to zero

Note on **`RANDOM`**:
Calling up `RANDOM( 0 )` delivers a random number between 0.0 and 1.0. `RANDOM` of a negative number is undefined. The command `COMPUTE xx = RANDOM( Max)` with a positive argument `MAX` delivers a whole random number in the range of 0 .. Max-1.

Generally simple (nuclear) variables are calculated. On the left side of the command, i.e. only **in front of** the indicating sign are variable families or variable groups also allowed. However the command then has a different meaning:

Assuming, G is a `DICHOQ`. Then `COMPUTE G = 1;` means that the first nuclear variable of G is set to `TRUE`; i.e. it contains the value which was defined when producing the variable family (e.g. the value 1 for "`EQ 1`"). This command is semantically identical to '`COMPUTE "G $1" = 1;`' if G was also defined as a `DICHOQ`, e.g.

```
DICHOQ G =
100 "text1"
*   "text2"
*   "text3"
*   "text4"
...........
...........
eq 1;
```

The command `COMPUTE G = var;` is the exact inverse of `SIMPLEVAR var = g;`

There is also a mechanism for deleting individual categories: if a command delivers a whole number within the range of the variable group which is also `MISSING` then the relevant category is deleted. How this is done is shown here:

```
COMPUTE del = 0;
MISSING del = 0;
COMPUTE G = 4 + del;
```

The command "`4+del`" is calculable because `del` contains a number; the command delivers the value 4. At the same time `del` counts as `MISSING` so that the result 4 is also `MISSING`. The above command is thus equivalent to "`COMPUTE 'G $4' = 0;`".

Variable families are treated accordingly. If F were a `MULTIQ` then `COMPUTE F = 2;` means that the first "free" nuclear variable of the `MULTIQ` F is searched for and given the value 2. A nuclear variable in a variable family counts as "free" if it is `MISSING`.

If the command after the indicating sign delivers a calculable numerical value with the characteristic `MISSING` (see above) then the variable family is searched until a nuclear variable contains this value; this is then deleted and replaced by the global `SYSTEMMISSING` (see `SETMISSING`).

If the command results in the value `SYSTEMMISSING` or a value in the `MISSING` list of the variable family then nothing happens.

COMPUTE COPY

Using COMPUTE COPY whole variable areas can be copied:

Example:

```
COMPUTE COPY v1 TO v27 = x1 TO x20, item1 TO item7;
```

These also offer the opportunity to allocate variables with multi-responses (VARGROUP, GROUPS, DICHOQs, VARFAMILIES and FAMILIYVARS) which cannot be used for arithmetical calculations. Instead of VARLIST v1 TO v27 above for example there could also be a VARGROUP of 27 individual variables (or columns).

If a filtered variable is to the right of the indicating symbol and the symbol is not valid then the SYSTEMMISSING value is stored to the left of the relevant variable (siehe SETMISSING).

COMPUTE SWAP

This is a COMPUTE COPY which works in both directions.

Example:

```
COMPUTE SWAP b1 b2 = a1 a2;
```

As a result the values in b1 and a1 and the values in b2 and a2 are swapped over.

COMPUTE LOAD

Syntax:

```
COMPUTE LOAD <zielvar> = <varlist> ;
```

zielvar: a SINGLEQ, VARFAMILY or VARGROUP
varlist: one or more SINGLEQ, VARFAMILY, VARGROUP or OPENVAR

Using COMPUTE LOAD the values and value sets can be displayed between different display formats. This is often used to display open questions (OPEN see above.) in a multi-response variable (VARGROUP or VARFAMILY), as in the following sequence for example:

```
SINGLEQ offen = TITLE "Offene Frage" OPEN;
MAKEFAMILY f_offen = 10;
COMPUTE LOAD f_offen = offen;
```

Afterwards the content of the open question can be manipulated as can any other VARFAMILY, e.g. for RECODE etc.

A VARFAMILY can however also be presented in a VARGROUP and vice versa, e.g. if an OUTFILE must be built using a rigid coding, e.g.:

```
MULTIQ ff = 10 2 labels
1 "Label 1"
...
30 "Label 30"
;
MAKEGROUP G = 30;
COPYLABELS g = f;
COMPUTE LOAD g = f;
```

LOAD and OVERCODE: if an OVERCODE is defined in a variable then using COMPUTE LOAD the OVERCODEs are carried over as synthetic values: a LOAD carries over all values which a variable would otherwise carry over to for example a table calculation part. COMPUTE COPY would only carry over the "real" values.

If COMPUTE LOAD has only one variable as an argument and the target variable does not yet exist then the target variable is produced with the same structure as the argument variable.

COMPUTE ASCEND
COMPUTE DESCEND
COMPUTE LOAD carries over the values in exactly the order in which they were found in the source. Using COMPUTE ASCEND or DESCEND the value sets are sorted in ascending or descending order before they are recorded in the target variable.

COMPUTE SHUFFLE
As COMPUTE LOAD only the values of the source variable are sorted randomly per variable. Is usually necessary for programming questionnaires. e.g.

```
COMPUTE SHUFFLE liste = oldliste;
```

Puts the values stored in <oldliste> in a random order into <liste>. If <liste> is a nuclear variable then a value can be selected from the set at random. The choice of "3 of 6" and the like is easy to achieve.
Example:

```
MAKEFAMILY neu = 3;
COMPUTE SHUFFLE neu = alt;
```

COMPUTE ADD
Syntax:

```
COMPUTE ADD <zielvar> = <varlist>;
```

The values from <varlist> are additionally stored in the variable <zielvar>. This obviously presupposes that <zielvar> is a multi-response variable (MULTIQ, DICHOQ).

COMPUTE SORT
Syntax:

```
COMPUTE SORT ( n1 n2 ... nm ) <zielvar> = <quellvar> ;
```

zielvar: a SINGLEQ, VARFAMILY or VARGROUP
quellvar: a SINGLEQ, VARFAMILY, VARGROUP or OPENVAR

COMPUTE SORT is a special variant of the COMPUTE LOAD construct. It can be the case with the presentation of a VARFAMILY that not all of the responses can be carried over, for example if a VARFAMILY only allows three responses but in the VARGROUP five items are "on" then there is overspill. After the three responses have been stored in the VARFAMILY the rest are ignored. Using COMPUTE SORT a priority rule can be laid down as to which values are to be stored.

```
VARFAMILY f = f.1 TO f.3;
COMPUTE SORT ( 2 3 6 7 1 0 ) f = g;
```

If the variable g contains the values 2, 4, 5, 7 and 10 then the 2 then the 7 are stored. The values 4, 5 and 10 are not on the priority list and are not stored.

COMPUTE ALPHA

Target and source must be ALPHA variables. The target variable is allocated a numerical value which conforms to the ALPHATEXT of the source variable, a new label is allocated if necessary. If the target variable does not yet exist then a new ALPHA variable is generated.

The argument for COMPUTE ALPHA may also be a string constant.

FCOMPUTE

Parallel to the COMPUTE statement there is also FCOMPUTE, which tests the filters set with SETFILTER. FCOMPUTE is only used if all the filter conditions are true or if there is no filter.

SIMPLEVAR

Syntax:

```
SIMPLEVAR <variable> = <vargroup> ;
```

Using SIMPLEVAR a variable which is coded in individual bits which is present as a variable group can be presented as a simple (nuclear) variable. Prerequisite: exactly one element in the group is "ON", all the others are "OFF".

If several elements are "ON", then the <variable> is MISSING (SystemMissing, value SETMISSING); if no elements are "ON" then the variable receives the value BLANKVALUE. Otherwise the <variable> receives a 1 if the first element is "ON" , a second a 2 if the second element is "ON" etc.

BCDVAR

Syntax:

```
BCDVAR <variable> = <vargroup> ;
```

The BCDVAR statement represents a variable group from a numerical variable. The variable group is interpreted as a binary coded decimal number: each set of 4 bits result in a decimal number; the values A ... F Hex are not permitted.

BITGROUP

Syntax:

```
BITGROUP <vargroup> = <varname> ;
```

The inverse of BCDVAR: the numerical value is broken down into its numbers and these are then broken down into the bits 2**0, 2**1, 2**2, 2**3. If the variable group is not large enough to take on all the binary numbers then the upper bits are not represented and there is no error message. The variable group must always be adequately dimensioned.

MEAN

Syntax:

```
MEAN <varname> = <Varlist>;
```

Allows the simple calculation of a mean of several variables within a case.
Example:

```
MEAN faktor = Item1 TO Item13;
```

Only those variables are included in the mean which are not MISSING in the current case. If all the variables are MISSING then the result is also MISSING.

SUM

Syntax:

```
SUM <varname> = <Varlist>;
```

Simplifies the calculation of a sum of several variables within a case.
Example:

```
SUM Summe = Item1 TO Item13;
```

Only those variables are included in the sum which are not `MISSING` in the current case. If all the variables are `MISSING` then the result is 0.

MIN

Syntax:

```
MIN <varname> = <Varlist>;
```

Produces the minimum of several variables within a case.
Example:

```
MIN minimun = Item1 TO Item13;
```

Only those variables are included which are not `MISSING` in the current case. If all the variables are `MISSING` then the result is `MISSING`.

MAX

Syntax:

```
MAX <varname> = <Varlist>;
```

Produces the maximum of several variables within a case.
Example:

```
MAX maximun = Item1 TO Item13;
```

Only those variables are included which are not `MISSING` in the current case. If all the variables are `MISSING` then the result is `MISSING`.

IF ... THEN ... ELSE ...

Allows the new calculation of variables under the control of logical conditions. The same syntax rules are valid for the `IF` clause as for `SELECT` and the description of the arithmetic syntax for `COMPUTE` is also valid for the `THEN or ELSE` part.
Example:

```
IF a GE 3 THEN
x = c * ( 4 - d )
ELSE
x = e / ( d + c );
```

The `ELSE` part of the command can be omitted, e.g. `IF a EQ 3 THEN d = 5;`

Compared with the set operator `IN` easily allows the test for the existence of values in multi-response variables. The test can look like this:

```
IF 4 IN famvar_01 THEN ...
```

A variable must always be on the right side. On the left side of the set test there can be a value, a value list, a nuclear variable or a multi-response variable. A test for number ranges would be as follows:

```
IF [ 1 : 13 ] IN Frage11 then ...
```

Number ranges can be mixed with individual values (as in RECODE). e.g.:

```
IF [ 1 3 55 : 60 -0.1 : 0.1 ] IN ...
```

Since an update (2009/10) variable lists in parenthesis may be on the right side of such comparisons with the IN operator.
Example:

```
IF [ 1 2 3 ] IN ( var1 var7 to var10 ) THEN ….
```

All variable types can be tested non-stop for MISSING using IN. There is a system test variable SYSMISS for this function.

The test SYSMISS IN <variable> has its own meaning for OPEN variables in CATI/CAPI programs: the test is always true if a response has been made to the open question.

The test "has no valid value" for example is thus:

```
IF SYSMISS IN <variable> THEN ...
```

The test "contains at least one valid value" for example is thus:

```
IF NOT ( SYSMISS IN <variable> ) THEN ...
```

If on the left side there is a multi-response variable then the test is TRUE if the intersection of the left and the right amounts is not empty (see also INIT for multi-response variables).
Example:

```
MAKEFAMILY test = 3;
INIT test = ( 4 66 19 );
IF test IN family THEN ....  ELSE .... ;
```

This means the same as:

```
IF [ 4 66 19 ] IN family THEN ... ELSE ... ;
```

and the same as:

```
IF 4 IN family
OR 66 IN family
OR 19 IN family THEN ....  ELSE  ..... :
```

The command "test IN family" is always TRUE if the variable family contains either the value 4 or 66 or 19.

NB: INIT is static and is only carried out during compilation of the program; i.e. the test variable (test in the example) may not be altered at another place in the script as it then no longer works.

The type of variable can also be tested in the antecedent of IF. This is usually only necessary in macros which may receive a formal parameter which contains any type of variable. The test has following syntax:

```
IF <varname> IS <vartype> THEN ... [ ELSE ... ] ;
```

The <vartype> can be:

```
VARIABLE SINGLEQ
FAMILYVAR MULTIQ
GROUPVAR DICHOQ
```

or

```
OPEN.
```

The terms on the same line have the same meaning.

The IF statement has the COPY flag analogue to the COMPUTE statement with which the complete variable list can be copied:
Example

```
IF a EQ x THEN
  COPY v1 TO v10 = x1 TO x10
ELSE
  COPY v1 TO v10 = y1 TO y10;
```

The same is valid for the LOAD flag (see also COMPUTE LOAD):
Example

```
IF a EQ x THEN
  LOAD fam = group1
ELSE
  LOAD fam = group2;
```

In IF statements also whole variable lists can be altered arithmetically. If a question filter for example is to be set explicitly then the following can be programmed (although the FILTER statement would obviously be more elegant):

```
IF Frage13 EQ "Nichtraucher" THEN Frage14 TO Frage23a = -1;
```

As can be seen from this example tests can be carried out for numerical and alphanumerical constants. Alphanumerical constants are placed in quotation marks (" or '). If alphanumerical constants are compared with variables the VALUELABEL of the current value are pulled into the comparison. For ALPHA variables this is the string currently being read and for other variables this is the label.
Example:

```
IF Partei EQ "SPD" THEN ...;
```

As soon as a string constant appears on one side of the comparison then the comparison as a text comparison is carried out. Greater Than and Less Than (GT or LT) for example are then carried out as lexical comparisons using the ASCII-Codes: "AA" is Less Than "aa", etc.

There can be Run-Time-Errors: if a variable is tested on a string and has a numerical value for which no label has been defined then the test cannot be determinable. ***GESS* tabs** then produces a Run-Time-Error.

NB: There is a syntactical problem: As variables must be placed in quotation marks if special variable names are used (e.g. variable names containing blanks) it cannot be syntactically definitively decided what is meant by a token in quotation marks in an IF condition. ***GESS* tabs** interprets strings in quotation marks or inverted commas as variable names if there is a variable with that name, otherwise it interprets them as text constants. Variables which are to be used in IF conditions should therefore be named with a simple token which need not be placed in quotation marks. The explanatory texts can be declared VARTITLE for the production of the tables.

FIF ... THEN ... ELSE ...;
Parallel to the IF statement explained above is the FIF statement which functions the same with one small difference: the FIF statement heeds the filters set with SETFILTER. If an FIF statement is between SETFILTER ... ENDFILTER then if the filter is not open (i.e. the filter condition is not true), neither the THEN nor the ELSE part is carried out.

VII Case Selection

There are several commands for case selection in tables: a permanent filter (SELECT), a table-related filter (TABSELECT), and finally TABLEFILTER. If more than one filter is used then only those cases are presented in the table which answer all the filter criteria. TABSELECT filters are preset and valid until the next table-related filter is set. An empty filter, i.e. a filter which contains all cases is thus:

```
TABSELECT;
```

SELECT
defines an import filter: only those cases which conform to this filter are processed further, i.e. all tables are produced only on the basis of this data; SELECT is a permanent filter as opposed to TABSELECT (see below). SELECT also acts on the output according to COPYFILE or SYSTEMOUT. An iterative weighting also only refers to the selected cases.

All RECODE, RANGES, COMPUTE or IF commands are carried out before SELECT; SELECT commands also refer to the end-product of recoding or re-calculated values.
Example:

```
SELECT alter GE 4;
SELECT Partei EQ 'CDU';
SELECT alter GE 4 AND Parteipräferenz EQ 1;
SELECT NOT ( alter EQ 2 und einkommen GE 3000 );
SELECT (alter EQ 2 OR alter EQ 3) AND (einkommen EQ 1 OR einkommen EQ
2);
```

The comparative operators allowed are EQ, NE, GT, GE, LE, LT and IN. For logical combinations there are AND, OR and NOT. Associations must be defined explicitly using parenthesis; rows of OR and AND without parenthesis are processed from left to right. The common abbreviations `a EQ 1 OR 2` instead of `a EQ 1 OR a EQ 2` etc. are not allowed. The IN test is for this purpose (see above IF ...). String constants are allowed.

TABSELECT
defines a selection of cases for the following tables. TABSELECT remains valid until a new TABSELECT is defined. Should all cases be processed in the following tables then simply:

```
TABSELECT;
```

is written. (This condition is always true.) The syntax equates to SELECT (non permanent filter). If a permanent filter (SELECT) is also defined then only those cases are evaluated which conform to the conditions (AND combination).

Using `TABSELECT TEXT <"Selektionsbeschreibung"> <Bedingung>;`
an explanatory text can be added to a table selection. This text is added to the TOPTEXT or BOTTOMTEXT in the following tables (implementation tip: ONLY for Postscript Output).

TABLEFILTER
Syntax:

```
TABLEFILTER <number> = TEXT "<text>" <condition>;
```

All table filters make an ARRAY (1...10) of selection conditions for the tabulation. They are valid as TABSELECT for the following table commands. All TABLEFILTER are connected with AND.

For Postscript output the texts are in TOPTEXT in the order of the addresses (1...10) if this is defined. If an empty text is defined ("") then the row is suppressed.

VIII Treatment of MISSING VALUES

For different reasons it is not always possible to collate all the variables in a questionnaire:

a) the interviewee did not answer the question posed, (the classic "no answer"), or
b) a question was not asked as it was not appropriate according to previous answers (the classic "not applicable").

Only in a) is the case MISSING in the exact sense; in case b) we prefer to speak of filtering.

A filtered variable is often presented as if the table has only been calculated on the basis of the cases which conform to the question filter. This then also affects the table-N. Technically this can be solved using the FILTER statement. The use of the ***GESS*** input program for Data-Entry steers filtering during input so that time is saved and errors avoided.

There are different strategies for presenting the real MISSING. A common strategy is to present all cases which are MISSING as "no answer" in the table. That goes in the direction of these values technically not being "MISSING VALUES" but rather inputting and evaluating them as a perfectly normal category. In as far as the variable is included as a numerical quantity in mean, variance etc. it is usually excluded from calculation.

This procedure however requires additional and error-prone effort when using multi-response variables. The Inputter/Coder must always additionally weigh up whether or not a variable should be noted as "no answer" or not. ***GESS* tabs** provides the additional mechanism AUTONOANSWER for this purpose: for all variables with this characteristic the system automatically generates a no-answer value if there is no valid variable value unless the variable filter conditions for the current case define it as not applicable. This is valid systematically for all variables whether they are nuclear variables or variables for multi-responses. This concept is partly built on the original MISSING concept: thus a simple (nuclear) variable for example is always a no answer of it is MISSING, a MULTIQ or VARFAMILY is always no answer if all responses included are MISSING. A VARGROUP or DICHOQ is always a no answer if it has no positive responses.

The data gathered from the ASCII file can therefore be seen as an "absent record" by there being a special MISSING symbol or the variable having a numerical value which has been defined as "absent". The latter occurs with the MISSING statement, the former with the MISSINGCHAR statement.

During output the MISSING values are marked with an "M" in the output file. If an "M" is found in the input file then the variable counts as MISSING without a MISSING value having to be declared. The value "M" can be redefined using the MISSINGCHAR statement.

This is all complicated enough but for the flexibility of tabulation there are additional rules governing when which case is to appear in which table. A summarised can be found under the heading USECASES in the chapter on tabulation commands and is recommended additional reading. The recommended strategy is summarised below:

1. Generally AUTONOANSWER to be used.
2. Either set questionnaire filter (SETFILTER/ENDFILTER) or explicitly filter table (TABSELECT)
3. set USESCASES to XANDYVALID.
4. set USEMISSING to NO.
5. data entry of absent values as MISSING (***GESS*** input does this automatically)

Example:

```
AUTONOANSWER = YES "K.A.";
USECASES = XANDYVALID;
USEMISSING = NO;
```

MISSING inheritance:

If variables are addressed by COMPUTE statements or the COMPUTE part of an IF command which has MISSING values defined for it then the result is also MISSING if the current value of one of the variables included in the calculation is MISSING. Also the variable receives the numerical value which is defined by SETMISSING (see below).

MISSING
Allows the definition of individual characteristics of nuclear variables as MISSING values.
Syntax:

```
MISSING <Varlist> = { number }*n;
(n <= 3)
```

SETMISSING
A MISSING value is automatically inherited on to variables which emanate from the calculation of other variables. If MISSING values go into a calculation or an 'M' is found in the input then the result is a MISSING value. The variable then receives the characteristic allocated by the user with SETMISSING.
Example:

```
SETMISSING = 9999;
```

USEMISSING
steers the evaluation of MISSING characteristics in TABLE and COMPARE.

```
USEMISSING = NO;
```

is the preset; using USEMISSING = YES; the MISSING values can be called up for the evaluation of the following tables.

MISSINGCHAR
defines the character used to mark MISSING values for input and output.
Preset: MISSINGCHAR = "M";

SETFILTER
ENDFILTER
FILTER
COPYFILTER
defines a variable (or list of variables) as filtered according to a condition: the filtered variables then never flow into the tables if the condition is FALSE. The most frequently used use is probably the filtering of questionnaires. This filtering can also be used to steer ***GESS* input**. SETFILTER is however also useful for limiting the valid range for VARGROUPS and VARFAMILY.

The syntactical constructs USEFILTER/MAKEFILTER have been retired as of Version 2.82. Earlier limitations of FILTER/SETFILTER/ENDFILTER in connection with RANDOM orders are no longer valid.

Using COPYFILTER a filter (perhaps more complex) can be inherited to recalculated variables. This is mainly useful for tabulation.

If a variable which has been filtered out (the conditions are FALSE) is referred to COMPUTE then the result is MISSING.
Syntax:

```
SETFILTER [ <filtername> ] [ TEXT "filtertext" ] = < log. Bedingung >
;
ENDFILTER [ <filtername> ] ;
COPYFILTER <varname> = <varname>;
```

Example:

```
SETFILTER Filter1 = Varxx EQ 2;
  SINGLEQ gefiltert = * labels
  1 ...
  ;
  SETFILTER Filter2 = gefiltert eq 1;
    SINGLEQ Doppeltgefiltert = * ...
  ENDFILTER;
  SINGLEQ WiederGefiltert = * ...
ENDFILTER Filter1;
```

SETFILTER and ENDFILTER administrate a filter batch. The currently valid filter is ended and possibly a previous filter is again valid using ENDFILTER. An ENDFILTER statement with an explicit name ends **ALL** filters until after the named filter if it exists. A named ENDFILTER can thus end more than one SETFILTER. If the named filter no longer exists in the batch then there is an error message.

If a SETFILTER is set within a SETFILTER – ENDFILTER Block then the conditions are combined using AND. The variable is thus doubly filtered with regard to content in the above example according to "gefiltert eq 1" as well as to "Varxx eq 2".

If a filter text is defined in SETFILTER the filter text is attached to the VARTEXT for all variables which are defined in the relevant filter block.
Example:

```
SETFILTER TEXT "Nur an Frauen" = geschlecht eq 1;
    SINGLEQ Grauanteil = text "
    Frage 4 (Kontakt):
    Wie hoch ist der Grauanteil in Ihrem Haar?
    " * labels
    ...
    ;
ENDFILTER;
```

If the tables were produced using CITEALLVARS or CITEVARTEXT the TOPTEXT for example would contain:

```
Frage 4 (Kontakt):
Wie hoch ist der Grauanteil in Ihrem Haar?

Nur an Frauen
```

Syntax:

```
FILTER <varlist> [ = <bedingung> | AS <varname> ] ;
```

Variables which have already been defined can be allocated filters using FILTER. If the indicating symbol follows the variable list a condition can be written behind it. If the key word AS follows it then the allocation of the same filter as for the other variables can be requested.

MENUFILTER

Syntax:

```
MENUFILTER <varlist> = [ YES | NO ];
```

In "TABLES BY MENU" (XGM and related) only those variables can be filtered which have been previously marked as filter variables by the originator of the *.TS-File. This occurs using MENUFILTER.

MENUMEAN

Syntax:

```
MENUMEAN <varlist> = [ YES | NO ];
```

In "TABLES BY MENU" (XGM and related) only those variables which have been previously marked as filter variables by the originator of the *.TS-File can be presented as mean, variance etc. This occurs using MENUMEAN.

MENUHEADER

Syntax:

```
MENUHEADER <varlist> = [ YES | NO ];
```

In XGMW (special form of XGM) only certain variables which have already been defined can be used in the stub. This are marked using MENUHEADER.

STATIC

Syntax:

```
STATIC <varlist> = [ YES | NO ];
```

When reading a new data set all variables are deleted. Using the characteristic

```
STATIC <varname> = YES;
```

a variable can keep its value.

This was the standard behaviour for all variables up until Version 2.91h. The previous standard behaviour can be forced using: AUTOCLEAR = NO;

AUTOCLEAR

Syntax:

```
AUTOCLEAR = [ YES | NO ];
```

New standard behaviour as of Version 2.91h is that all variables are set to MISSING before reading. The previous behaviour can be resumed in this way.

ASALPHA

Syntax:

```
ASALPHA <varlist> = [ YES | NO ];
```

Is only valid for OPEN variables. Ultimately a code is allocated to every open text and the open texts are assigned to the variables as VALUELABELS. The same texts have the same codes. In other words an open question is treated as an ALPHA variable.

AUTONOANSWER

Syntax:

```
AUTONOANSWER [ <varlist> ] = [ YES "noanswertext" | NO ] [ LEVEL <
number > ;
```

AUTONOANSWER is either (without <varlist>) preset or it refers to explicit variables and the preset remains unchanged for all further variables.

In this way consistent behaviour for MISSING VALUES can be achieved regardless of the variable type (SINGLEQ, DICHOQ, VARGROUP, MULTIQ or OPENVAR). An additional variable label with the "no answer" text is automatically produced. Also a logical switch is set so that the variable delivers the no answer value should there be no valid value.

The preset is valid for all variable/label generation after the statement up until the next AUTONOANSWER statement.

With the level components a COUNTLEVEL for the AUTONOANSWER category can be set. A LEVEL <> 0 indicates that the frequency value is not to be included in the TOTAL (see MULTITOTALX, MULTITOTALY).

IX Weighting

***GESS* tabs** also processes weighted data sets in all tables. Weights which are included in the data set can be introduced from external sources using the WEIGHT statement. Weights can also be calculated internally per case and extracted from a variable. This also occurs using the WEIGHT statement.

In many cases it is easier and more flexible to stipulate one or more marginal distributions for ***GESS* tabs**. The stipulation of the rated value occurs using the WEIGHTCELLS statement. As soon as at least one WEIGHTCELLS command is in the leading file ***GESS* tabs** executes an iterative factorial weighting. In the standard case ***GESS* tabs** begins with the weighting factor 1.0 per case. A starting value can also be defined using the WEIGHT command which ***GESS* tabs** extracts from the data set. The absolute value of the natural logarithm of the greatest deviance between target and actual value regarding all the stipulated cells serves as the abort criterion. The required accuracy can be defined using WEIGHTACCURACY commands. If the target distribution is not achieved after 50 iterations then the program is aborted. If a SELECT command has also been given then only the selected cases are weighted, i.e. the weighting prerequisites only refer to the selected subamount of cases.

Using the parameters MAXIMUMWEIGHT and MINIMUMWEIGHT the value range allowed for the calculated weights is defined. Using MAXIMUMWFACT and MINIMUMWFACT the adaptation can be lessened, they define the maximum alteration per iteration step e.g. if the maximum or minimum is reached with 1.05 or 0.95 that the weight per step alters a maximum of 5% upwards or downwards.

WEIGHT
discloses where an externally calculated weight is in the data set:
Example:

```
WEIGHT = 62 6;
```

(weight is in column 62, Len=6)

Alternatively a known variable can be named:

```
...
COMPUTE gewicht = ( a + c ) * 0.1;
WEIGHT = Gewicht;
...
```

Here it should be noted that the command WEIGHT= is carried out in the RunTime-Module directly before the case is fed into the tables i.e. after all COMPUTE, IF or RECODE statements have been processed. The variable to be used for weighting should therefore not be modified after the WEIGHT statement.

USEWEIGHT
Syntax:

```
USEWEIGHT = [ YES | NO | <varname> ] ;
```

To check the weight calculation it can be interesting to analyse the system weight unweighted in SYSTEMWEIGHT. Then it is interesting to import the weight but not to use it. USEWEIGHT can be set to NO for such purposes.

The nuclear variable name can also be quoted: this variable's contents is then valid for all the following tables (until the next USEWEIGHT statement) as the current weight. Tables can also be built with differently weighted cells/rows or columns using TABLE ADD.

WEIGHTOUT
defines where a newly calculated weight is to be stored in the outfile. Syntax as above.
Example:

```
WEIGHTOUT = 68 6;
```

WEIGHTCELLS

Requests weighting according to the variable characteristics. As soon as at least one WEIGHTCELLS statement is found in the source text the program carries out an additional reading run of the data in which the weight factors are calculated. If there are more than one WEIGHTCELLS statement present iterative weighting continues until all the weighting conditions have been fulfilled. If a consistent weighting is not possible the program run is aborted.

Syntax:

```
WEIGHTCELLS [ AUTOALIGN ] <varname> =
{ <code> : <sollwert> % }*n
[ MISSING : <code> : <sollwert> %
;
```

Example:

```
WEIGHTCELLS Geschlecht =
1 : 48%
2 : 52%
MISSING : 12% ;
```

The sum of all percentages must come to 100%. The MISSING clause is optional; if it is absent there must be a specification for all empirically present cells. If a weight is defined from the input (see WEIGHT), then this weight is used as the starting weight for all cases.

If 0.0 is defined as the percentage for MISSING then the empirically present percentage is documented for MISSING, i.e. in an additional run through the data it is established how many cases do not conform to the designated weighting cells and their percentaged share is inserted. As for the weighting, a compacted version of the data is stored in ram anyway so this tally is of little import and hardly affects performance.

If weighting is to occur according to a combination of several variables then the relevant combination variable is to be generated.

Example:

```
COMPUTE altsex = alter * 10 + sex;
WEIGHTCELLS AltSex =
11 : 25%
12 : 25%
21 : 25%
22 : 25%
;
```

Instead of the simple percentage the output in the form <constant> / <constant> % is also allowed.

The sum of the percentaged prerequisites must come to 100%. If this is not the case then the evaluation run is aborted with an error message. Alternatively the key word AUTOALIGN can also be used to cause a proportional conversion to 100%.

WEIGHTSUM

States the desired sum of the weights to be calculated. Normally weighting occurs to the number of the cases physically read.

WEIGHTACCURACY

Defines the accuracy bound up to which iteration should occur. WEIGHTACCURACY is the natural logarithm of the maximum deviance of a weighting cell from the prerequisite as factor.

Preset: WEIGHTACCURACY = 0.0001;

MINIMUMWEIGHT
MINIMUMWFACT
MAXIMUMWEIGHT
MAXIMUMWFACT
Preset for the control of weighting. MINIMUMWEIGHT and MAXIMUMWEIGHT set the minimum or maximum weight of a case. MINIMUMWFACT and MAXIMUMWFACT set a limit for the factorial alteration of the weight per iteration cycle.

Preset:

```
MAXIMUMWEIGHT = 1E+20;
MINIMUMWEIGHT = 0;
MAXIMUMWFACT = 1E+20;
MINIMUMWFACT = 0;
```

(usually no limitations)

X Controlling Table Appearance

The output of values and texts in tables can be adapted to individual wishes using a variety of options. There are decimal options (DECIMALS, SETDECIMALS), label options (PRINTALL, GLOBALPRINTALL), text options (DOCUMENT, CASESTITLE, NOMINATIONTITLE, TABLETITLE, TABLENUMBER, TOTALTITLE, PHYSICALNTITLE, MULTISTRING, CASEBASESTRING, RAWDATASTRING) and table or page dimension options (UNITS, PAPER, MARGINS, MINTABLEWIDTH, MINCOLWIDTH, MINLABELWIDTH, MAXLABELWIDTH, MINFRAMECOLWIDTH).

Decimal Options

DECIMALPERCENT
Defines the number of decimal places for the percentages in cross tables (TABLE) or comparative tables (COMPARE). DECIMALPERCENT settings are valid for all following tables until the next DECIMALPERCENT command.
Example:

```
DECIMALPERCENT = 1;
```

Preset: `DECIMALPERCENT = 0;`

DECIMALS
The number of decimal places can be stipulated for variable output if no VALUELABEL has been allocated and PRINTALL=YES. It is also used for MEAN or SUM output.
Example:

```
DECIMALS = 2;
```

The characteristics or rather sums or mean of all variables then defined have two decimal places after the comma. This is valid until the next DECIMALS command (see also: DECIMALPERCENT for controlling the output of the decimal places for percentages).

Preset: `DECIMALS = 0;`

SETDECIMALS
Serves to explicitly set the decimal point for variables which have already been defined.
Syntax:

```
SETDECIMALS < Varlist > = number ;
```

Label Options

PRINTALL
GLOBALPRINTALL
These options steer the output of unlabelled values. Usually unlabelled values are printed with a label generated from the numerical value.

It can however be required to suppress outliers in the tables: unlabelled values are to be treated as outliers where necessary and not be printed. This is achieved using PRINTALL = NO or GLOBALPRINTALL = NO. The latter as a preset refers to all variables which are defined after it. It thus changes the preset to PRINTALL. PRINTALL refers to an explicit variable list or the variable to have been defined last.
Example:

```
GLOBALPRINTALL = YES;
```

It is valid until the next GLOBALPRINTALL command.

Options for Table Dimensions:

This is where the field of differences between Postscript printers and other printers begins. ***GESS* tabs** considers all printers which cannot produce postscript as simple line printers without proportional fonts. This classification is admittedly somewhat coarse as much of what ***GESS* tabs** uses postscript for is also possible with e.g. HP PCL but just not everything. At today's hardware pricing there is however no real argument against a Postscript compatible printer.

A cohesive example for Postscript:**(PS)**

```
units = mm;
paper = height 297 width 210;
margins = left 20 right 20 top 20 bottom 22;
maxlabelwidth = 32;
minlabelwidth = 20;
mincolwidth = 19;
minframecolwidth = 13;
mintablewidth = 160;
units = points;
```

Comment:
***GESS* tabs** uses with Postscript in accordance with Postscripts graphic model the typographical point as its standard measurement. This is stated using the UNIT setting UNITS = POINTS at the end; in order to make the paper description somewhat easier for normal Middle Europeans there is a changeover to millimetre.

UNITS
Syntax:

```
UNITS = [ MM | POINTS | INCH ];
```

Internally ***GESS* tabs** works with typographical points or with characters or rows (1/72 inches) depending on the chosen printer model. Using UNITS the unit of measurement taken as a basis can be defined for the following dimensional commands:

```
PAPER
MARGINS
MAXLABELWIDTH
MINLABELWIDTH
MINCOLWIDTH
MINFRAMECOLWIDTH
MINTABLEWIDTH
MINTABLEHEIGHT
```

For line-orientated printers the width of the letters or the number of rows must be defined as the basic unit of measurement. The following syntax is valid:

```
UNITS = CPI <number> LPI <number>;
```

CPI means Characters Per Inch (Pitch); LPI means Lines Per Inch.
Example:

```
UNITS = CPI 10 LPI 6;
```

During output with USEFONT ***GESS* tabs** ensures that the fonts match the chosen settings for UNITS.

ELASTICITY (PS)
Elasticity is a measurement of how the scaling in the X direction is allowed to differ from the scaling in the Y direction. Preset: `ELASTICITY = 0.15;`

Background: Printing in Postscript offers the possibility to scale tables to fit which are larger than the available area on a page. This adjustment can be made independently in the X or the Y direction. This can lead to the fonts also being proportionally altered and a letter may stay the same height but become wider or remain the same width but become shorter. The elasticity defines the extent to which the X-Axis can be compressed independently to the compression of the Y-Axis. Using the elasticity setting of 0.0 the X-Axis and the Y-Axis would be compressed equally, i.e. the letters keep their proportions. The setting 1.0 allows the compression of the X or Y-Axis independently of the other.

PAPER
Syntax:

```
PAPER = HEIGHT <number> WIDTH <number>;
```

Example:

```
PAPER = HEIGHT 297 WIDTH 210;
```

An A4 page is 297mm long and 210mm wide. The interpretation of `<number>` complies with `UNITS`.

The following paper sizes are the usual standards (in typographical points):

PageSize Letter:	[612 792]
PageSize Legal:	[612 1008]
PageSize A0:	[2384 3370]
PageSize A1:	[1684 2384]
PageSize A2:	[1191 1684]
PageSize A3:	[842 1191]
PageSize A4:	[595 842]
PageSize A5:	[420 595]
PageSize A6:	[297 420]
PageSize A7:	[210 297]

MARGINS
Syntax:

```
MARGINS = LEFT <number> RIGHT <number> TOP <number> BOTTOM <number> ;
```

The gaps from the edge of the paper are defined using `MARGINS`. A border of 20 bis 25mm is usually used for A4.
Example:

```
MARGINS = LEFT 20 RIGHT 20 TOP 20 BOTTOM 20;
```

MAXLABELWIDTH
MINLABELWIDTH (PS)
MINCOLWIDTH
MINFRAMECOLWIDTH
MINTABLEWIDTH
MINTEXTHEIGHT(PS)
MINTABLEHEIGHT(PS)
These are used to define the basic size of the table: the maximum and minimum width of the label text in the banner, the minimum width of the data columns, the minimum width of the frame columns (Total, Absolute, unweighted, n), the minimum width and height of the table. The `UNITS` setting also has an affect on these settings.

`MINLABELWIDTH` and `MINTABLEHEIGHT` are ignored when printing with a line printer.

MINTABLEHEIGHT leads to an even increase in all LABELS Y cells.

Text Options:

HEADER
FOOTER
Syntax:

```
FOOTER = "text" [ LEFT | HCENTER | RIGHT | ] ;
```

A FOOTER statement sets a footer and HEADER a header. The rows are printed above or below the TOP or BOTTOM MARGINS. If LEFT or RIGHT options are given then the text is justified to the left or the right margins. HCENTER (preset) centres the table between the left and right margin.

If a NUMBERCHAR is found in the text then it is replaced with the current page number. The current page number can be changed using the PAGENUMBER statement.

The fonts for FOOTER and HEADER can be set using USEFONT FOOTER = ...;

PAGENUMBER
Resets the current page number.

INSTITUTION
Specifies the printing of the name of the institute added at the bottom left edge. The valid text is expanded to the right. Repeated use of the INSTITUTION statements can lead to meaningless results.

(PS): this text can have more than one line in output from Postscript printers with the backslash marking the end of a row.

DOCUMENT
Specifies a document indicatorwhich appears at the bottom right under the tables.
Example:

```
DOCUMENT = "Demo 2009";
```

The key words DATE and/or TIME produce a date or time. TIME and DATE key words can be mixed with any number of strings.
Example:

```
DOCUMENT = "Auszählung vom" DATE "    Zwischenstand" TIME;
```

Valid for all tables.

PS): a document indicator can have more than one line in output from Postscript printers with the backslash marking the end of a row.

DATEFORMAT

```
DATEFORMAT = <string>;
```

In the string the letters Y, M and D are expanded to year, month and day. All other symbols are taken into the date.

Thus:
DATEFORMAT = "dd.mm.yyyy"; results in the standard European date: 31.10.2009

`DATEFORMAT = "mm-dd-yy";` results in the American date: 10-31-09

This influences the output of the date in `DOCUMENT`.

DESCRIPTION
Example:

```
DESCRIPTION MEAN = Mittel;
```

Usually an explanation of the cell content is printed top left when using `TABLE` and there are standard texts for this in the system. If these texts are to be altered then the `DESCRIPTION` command is used, otherwise the texts can be switched off using `TABLEFORMAT = NODESCRIPTION;`

DESCRIPTIONSTRING
Alternatively a descriptive text can be explicitly set.
Example:

```
DESCRIPTIONSTRING = "Mittelwert|Absolut";
```

Individual rows are separated using a vertical line.

CASESTITLE
If the standard text "number of Interviewees abs." in `TABLEBASE = CASES` is to be replaced it can be done in this way:

```
CASESTITLE = "Number of Inter-views (abs.)";
```

The `CASESTITLE` can be set differently for the X and Y axes
Example:

```
CASESTITLE X = "n";
CASESTITLE Y = "N";
```

The same separating rules are valid as for `VALUELABELS` and are valid for all tables until changed.

NOMINATIONTITLE
In `TABLEBASE = NOMINATIONS` the standard text is "No. of responses abs.". This can be replaced.
Example:

```
NOMINATIONTITLE = "Nennungen";
```

Different texts are possible for the X and Y axes analogue to `CASESTITLE` (see above). It is valid for all tables until changed.

TABLETITLE
If the standard text "Table #:" is to be replaced it can be done as follows:

```
TABLETITLE = "Summary Table";
```

If the test is not to appear at all, then:

```
TABLETITLE = "";
```

If the program finds a hash "#" (more precisely: the `NUMBERCHAR`) in the string this character is replaced by the current table number. This is valid for all tables until it is changed.

LINEFEEDCHAR
NUMBERCHAR
SPLITCHAR
SPLITCHARSTAY
Preset:

```
Linefeedchar: \  Numberchar: #
Splitchar: -     Splitcharstay: #
```

Certain symbols have a special meaning for string output. The LINEFEEDCHAR causes a return in labels or variable titles. The NUMBERCHAR is replaced in table titles by the current table number. The SPLITCHAR allows a break in a word; the SPLITCHARSTAY also allows a break in a word but is printed as a hyphen even if it is not at the end of a line.

These symbols can be redefined but it should be noted that system standard texts may also have to be changed e.g. the TOTALTITLE: "Ins-ge-samt". The minus sign as a hyphen would also have to be changed if the SPLITCHAR were changed.

The current setting for these special characters is stored in SYSTEMOUT and where necessary re-established in SYSTEMIN.

TABLENUMBER
Defines the first number for the tables. Preset: TABLENUMBER = 1;

All tables share the same number range. The tables are only counted if there is a hash in the TABLETITLE. This is valid for all tables until changed.

TOTALTITLE
If the standard text "Insgesamt" is to be replaced then:

```
TOTALTITLE = Total;
```

The TOTALTITLE can be set differently for the X or Y axes:
Example:

```
TOTALTITLE X = "Total";
TOTALTITLE Y = "Insgesamt";
```

This is valid for all tables until changed.

PHYSICALNTITLE
Serves to replace the standard text "unweighted" with the indicator of columns or rows with "unweighted n". (See FRAMEELEMENTS, PHYSICALROW or PHYSICALCOLUMN).
Example:

```
PHYSICALNTITLE = "Zahl der Be-frag-ten";
```

X and Y frame texts can be set differently analogue to TOTALTITLE. This is valid for all tables until changed.

MULTISTRING
Defines the text in CODEBOOKs which refers to possible multi-responses.
Preset: MULTISTRING= "Mehrfachnennungen möglich";

This is valid for all tables until changed.

CASEBASESTRING
Defines the text which refers to the percentaging for multi-responses CODEBOOK tables.
Preset: CASEBASESTRING = "Prozentuiert auf die Zahl der Fälle";

This is valid for all tables until changed.

RAWDATASTRING
Indicator to differentiate between unweighted and weighted tables. Comes directly before DOCUMENT.
Preset: RAWDATASTRING = "*";

This is valid for all tables until changed.

Options for Printing and Layout of Tables

COMPRESSCODEBOOK

```
COMPRESSCODEBOOK = [ YES | NO ];
```

In the ASCII mode a list of CODEBOOKS can also be printed in a compressed form where a number of CODEBOOKS fit on to one page.

DRAWBOX
(PS): is ignored by line printers.
Syntax:

```
DRAWBOX <boxname> =
<number> { [ TOP | LEFT | RIGHT | BOTTOM | BOXRADIUS <number> ] }*n ;
```

All table elements are in BOXES in Postscript printout and all these boxes can have a border. Each edge of the box can be referred to individually using the key words TOP, BOTTOM, LEFT, RIGHT, and the thickness of each border can be defined in typographical points (1 point = 1/72 inch, approx. 0.3mm). Additionally the boxes can have a border with rounded corners using the key word BOXRADIUS by stipulating the radius of a circle at the corners in typographical points. If a BOXRADIUS is defined it is automatically valid for the edges of a box.
Example:

```
DRAWBOX TABLE = 1.0;
DRAWBOX DATACELL = 0.1 LEFT RIGHT;
DRAWBOX TABLE = 1 BOXRADIUS 10;
```

The following key words are recognised for boxes:

LABELS X	VALUELABELS on the X-Axis
LABELS Y	VALUELABELS on the Y-Axis
VARTITLE X	variable title on the X-Axis
VARTITLE Y	variable title on the Y-Axis
LABELSET X	box around VARTITLE and LABELS on the X-Axis
LABELSET Y	box around VARTITLE and LABELS on the Y-Axis
FRAMETITLE X	box around the FRAMEELEMENTS identifier on the X-Axis, e.g. Total.
FRAMETITLE Y	box around the FRAMEELEMENTS identifier on the Y-Axis, e.g. Total.

FRAMETITLEBOX X	box around all FRAMETITLE boxes on the X-Axis
FRAMETITLEBOX Y	box around all FRAMETITLE boxes on the Y-Axis
FRAMECELL X	box around the individual data elements in the frame columns (elements of the X-Axis)
FRAMECELL Y	box around the individual data elements in the frame rows (elements of the Y-Axis)
FRAMEBOX X	box around all FRAMECELL X
FRAMEBOX Y	box around all FRAMECELL Y
FRAMECROSS	the point of intersection of FRAMEBOX X and FRAMEBOX Y.
DATACELL	each individual cell of the table
DATABOX	box around all DATACELLS which belong to the crossing of two variables at one time.
TABLETITLE	box around the TABLETITLE
TOPTEXT	box around the TOPTEXT
BOTTOMTEXT	box around the BOTTOMTEXT
DOCUMENT	box around the DOCUMENT identifier. A small one line box below (or rather outside of) the table
DESCRIPTION	box around the description of the data cells contents in a table
STATISTICS	box around the statistic identifier below the relevant DATABOX
GRAPHBOX	box with the line graphics in PROFILE tables
TABLE	the whole table is really a BOX

BOXMINHEIGHT
(PS): is ignored by line printers.
Syntax:

```
BOXMINHEIGHT <boxname> = <number> ;
```

Height of the box in points. As the boxes are dependent on another height is not evaluated by all <boxnames>, e.g. the height of the DATACELL orientates itself to the minimum height which is defined by the LABELS X. However is valid for:

```
TABLETITLE
TOPTEXT
BOTTOMTEXT
VARTITLE X or Y
LABELS X or Y
FRAMETITLE X or Y
```

BOXLINEFEED
(PS): is ignored by line printers.
Syntax:

```
BOXLINEFEED <boxname> = <number> ;
```

Defines the gap between the rows within a BOX. Valid as above (see BOXMINHEIGHT).

SHADE
(PS): is ignored by line printers.
Syntax:

```
SHADE <boxname> = <number> ;
```

All table boxes can also be shaded grey using the SHADE command with which the shade of grey is stipulated for the Postcript Interpreter. White is 1.0 and black is 0.0 and ideal values for grey are between 0.75 and 0.99. All box names are valid as described above for DRAWBOX.

BACKGROUND
Syntax:

```
BACKGROUND <boxname> = <hue> <saturation> <brightness>;
```

or

```
BACKGROUND <boxname> = <Red> <Green> <Blue>;
```

Using the BACKGROUND command the background can be coloured. Usually the colour is stipulated using the Hue-Saturation-Brightness model. Hue, Saturation and Brightness are three parameters which all have to lie between 0.0 and 1.0. Hue has a range of colours arranged in a circle, the value 0.0 (as well as 1.0) is red, 0.33 is green and 0.67 is blue; in-between are all the other shades, yellow for example being 0.12 (between red and green), violet is 0.8 (between blue and red) etc. Saturation starts with 1.0 as the pure colour, saturation 0.0 is pure grey. Brightness at 0.0 is always black, at 1.0 is the maximum brilliancy.
Alternatively the colours can be chosen according to the RGB model by mixing red, green and blue. The parameter RGB = YES has to be set for this (see below).

FOREGROUND
Analogue to BACKGROUND and is used to shade the foreground which normally means the colour of the text.

FRAMECOLOR
The colour of the frames can also be defined using HSB or RGB as above.

COLOR FOREGROUND
or
COLOR BACKGROUND
With the COLOR statement DATACELLS and FRAMECELLS can be coloured depending on the value, e.g. all mean above a certain value are printed in red etc.
Syntax:

```
COLOR [ FOREGROUND | BACKGROUND ] =
{ |
[ DATABOX <number> <number> CODE [ X | Y ] <number > ]
<cellelement> RANGE <low> <high> = <number> <number> number> }*n
;
```

The definition of the colours is achieved again using either RGB or HSB (see above). Cells can only be coloured according to a certain value, e.g. MEAN is only coloured if a MEAN is contained in the CELLELEMENT. If a cell contains more than one cell element then all the rows are coloured due to the COLOR FOREGROUND statement.
Example:

```
COLOR BACKGROUND =
| MEAN RANGE 0 20      : 0.9 0.5 0.7
```

```
| MEAN RANGE 150 9999 : 0.4 .05 1.0
;
```

Example:

```
COLOR BACKGROUND =
| DATABOX 1 1 CODE X 1 MEAN RANGE 0 20     : 0.9 0.5 0.7
| DATABOX 1 1 CODE X 1 MEAN RANGE 150 9999 : 0.4 .05 1.0
;
```

Using the DATABOX commands individual columns or rows can be referred to in the table.

COLOR FOREGROUND behaves slightly differently to COLOR BACKGROUND since 2007: the background is always coloured for the whole cell because the individual cell elements have not been allocated an unambiguous area of background. COLOR FOREGROUND always refers to the text in the cell elements which can be unambiguously referred to and thus separately coloured.

Synthetic numerical values for indicators of significance:
COLOR FOREGROUND or BACKGROUND refer to the steering of numerical values. Usually this is what is required but unfortunately it does not work for colouring the letters showing significance. For this reason there is a synthetic or virtual numerical value of 99999 if the test in a cell is significant. If for example in COLCHIQU the test is for the range of 99998.9 - 99999.1 then it is always true if one of the required significance levels has been reached.

RGB

Syntax:

```
RGB = [ YES | NO ];
```

If RGB = NO ***GESS* tabs** calculates the numerical colour information according to the HSB model. If RGB = YES the numerical values are interpreted according to the Red-Green-Blue model.

Differing from the "usual" Windows standard ***GESS* tabs** uses the RGB notation from Adobe. Typically the pigment content is presented as an integer in the range of 0 ... 255; this gives a colour space of a total of 24 bits. For professional printing this is not sufficient and more to the point too inflexible. Adobe uses numerical values from one to ten for each colour which can be stipulated with as much accuracy as necessary. ***GESS* tabs** has adopted this superior colour model.

In **GESStabs.exe** under the menu "Options" there is the Windows-Colorpicker. If this is used **GESStabs.exe** writes the RGB value as a triple of the decimal numbers in the Defines-Textfield on the user interface. From there the selected colour can easily be copied into the script.

ALIGN

(PS): is ignored by line printers.
The text in each box can be positioned vertically as well as horizontally. The following terms are required: TOP - VCENTER - BOTTOM and LEFT - HCENTER - RIGHT.

Example:

```
ALIGN LABELS X = HCENTER VCENTER;
```

The terms LEFT and RIGHT can also contain a command for the distance to the edge of the box:

```
ALIGN LABELS Y = LEFT 5 VCENTER;
```

or

```
ALIGN DATACELL = VCENTER RIGHT 6;
```

If text elements are to be centred horizontally between other lines rather than the left and right edge then this can be achieved like this:

```
ALIGN TOPTEXT = RIGHT 20 LEFT 5 HCENTER TOP;
```

The list of box names listed above under DRAWBOX is also valid here. Some boxes never contain text e.g. LABELSET X. The ALIGN settings are thus never evaluated but they can be stipulated.

For texts (i.e. VARTEXT, TOPTEXT, BOTTOMTEXT) an ALIGN statement can set a tabulator:

```
ALIGN TOPTEXT VCENTER LEFT 6 TABULATOR 55;
```

The above TABULATOR key word defines a tabulator with an interval of 55 typographical points to the left edge. In the text itself a tabulator can be set within a row using the TAB key (ASCII 9).

If the table ("TABLE") is made an argument of the ALIGN statement, e.g. ALIGN TABLE = HCENTER VCENTER; the table itself can be positioned on the paper. In the above example it is vertically centred between TOP and BOTTOM (see PAPER statement) and horizontally between LEFT and RIGHT. The additional stipulations for the space between LEFT and RIGHT are ignored in the ALIGN statement.

Centring can only occur if the dimensions of the table are smaller than the space available according to the specifications of PAPER and MARGINS. If the table is larger than the available space then it will be squashed into the space specified by PAPER and MARGINS. Here the X and the Y dimension are treated separately. This is a good solution as long as the text is not too small and the squashing of X and Y are not too different otherwise the text will look too flat or too thin. Should this be the case a division of the table should be considered.

DISTANCE
(PS): is ignored by line printers.
Usually there are no gaps between the different boxes which make up the table. Spaces can however be defined in the X and the Y direction.
Example:

```
DISTANCE INTERBOX X = 13;
DISTANCE INTERBOX Y = 13;
```

In the standard form (see above) all spaces are set to the stipulated value. DISTANCE INTERBOX can also be differentiated:

Valid for X: (horizontal space)

```
DISTANCE INTERBOX X 1 = <Zahl>;  Distance of label boxes Y to left edge
DISTANCE INTERBOX X 2 = <Zahl>; Distance to total boxes
DISTANCE INTERBOX X 3 = <Zahl>; Distance to first variable Y
DISTANCE INTERBOX X 4 = <Zahl>; Distance between several variable blocks
DISTANCE INTERBOX X 5 = <Zahl>; Distance of right edge to last variable block
DISTANCE INTERBOX X 6 = <Zahl>; Left and right distance from TOPTEXT and BOTTOMTEXT to
                                edge
```

Valid for Y: (vertical space)

```
DISTANCE INTERBOX Y 1 = <Zahl>; Distance to TOPTEXT
DISTANCE INTERBOX Y 2 = <Zahl>; Distance of the labels X to TOPTEXT
DISTANCE INTERBOX Y 3 = <Zahl>; Distance of TOTAL row to labels
DISTANCE INTERBOX Y 4 = <Zahl>; Distance of first data block
DISTANCE INTERBOX Y 5 = <Zahl>; Distance between several data blocks
DISTANCE INTERBOX Y 6 = <Zahl>; Distance to BOTTOMTEXT
DISTANCE INTERBOX Y 7 = <Zahl>; Distance to lower table edge
```

The parameter for all distances is not evaluated if the relevant object is not present. Distances are evaluated in Postscript points (1/72 inch).

There is also the form DISTANCE INTERCELL with which a vertical minimum space between the data cells is defined.
Example:

```
DISTANCE INTERCELL = 6;
```

LINEFEEDFACTOR (PS)
Syntax:

```
LINEFEEDFACTOR = <number>;
```

Specifies the interlinear space and is preset to 1.15, i.e. 15% interlinear space, and should not be lower than 1.0.

USEFONT
(PS): Syntax: `USEFONT <Zielname> = <Fontname> SIZE <number>;`
(NON-PS): Syntax: `USEFONT <Zielname> = <Fontname>;`

This stipulates the standard font and binds special fonts to particular cell contents or formal table parts. Should the font generally be in Helvetica 10 point, the mean e.g. in Courier 12 point:

```
USEFONT "Helvetica" SIZE 10;          {standard font}
USEFONT MEAN = "Courier" SIZE 12;     {special font for MEAN}
```

It is very important to stipulate the standard font as soon as possible and this should take place in the case of commands directly after the PRINTFILE statement and definitely before the first VALUELABELS or VARIABLE statements with LABELS clause. The standard font is set using a USEFONT statement with an empty target name.

(PS): The standard for Postscript printers regarding the available fonts is the APPLE LaserWriter. This printer, and indeed the majority of the common postscript compatible laser printers have eight font-families in four variations (normal, italics, bold and bold italics) at their disposal plus three special fonts; symbols, dingbats (all sorts of little pictures), and a cursive script. TABLES recognises the following font names used in printers compatible with Laserwriter:

Courier *Courier-Oblique* **Courier-Bold** ***Courier-BoldOblique***

Helvetica-Narrow **Helvetica-Narrow-Bold**
Helvetica-Narrow-Oblique ***Helvetica-Narrow-BoldOblique***
Helvetica **Helvetica-Bold** ***Helvetica-BoldOblique*** *Helvetica-Oblique*

AvantGarde-Book *AvantGarde-BookOblique*
AvantGarde-Demi ***AvantGarde-DemiOblique***

Times-Roman *Times-Italic* ***Times-BoldItalic*** **Times-Bold**

Bookman-Demi ***Bookman-DemiItalight***
Bookman-Light *Bookman-LightItalic*

NewCenturySchlbk-Roman *NewCenturySchlbk-Italic*
NewCenturySchlbk-Bold ***NewCenturySchlbk-BoldItalic***

Palatino-Roman *Palatino-Italic*
Palatino-Bold ***Palatino-BoldItalic***

Symbol (Symbol)

ZapfChancery-MediumItalic

ZapfDingbats (ZapfDingbats)

The spelling of these font names is to be copied exactly, the capital letters are also important. The standard font meter (POSTSCR.TBL) for ***GESS* tabs** uses these fonts as a basis.

This method of storing the necessary information in the postscr.tbl file to the most common ADOBE scripts for ***GESS* tabs** is sufficient for most uses and is easy to use. Problems arise if other fonts are used or if more "exotic" encodings are required as ***GESS*** has adopted this method from the time of DOS. Due to this connection with DOS only those characters can be printed faultlessly which exist both in the LATIN1 character set (often known as ANSI) and also in the DOS character set IBM850. However this falls at the first hurdle, namely with the € as this obviously came after DOS.

There is an additional method of introducing the system to scripts and font meters: the ADOBE AFM-Files (Adobe Font Metric). All the fonts that are to be used have to be stored in a directory in the AFM File. The information is passed on to ***GESS* tabs** by means of an environment variable called GAFM. If this environment variable is set ***GESS* tabs** interprets all AFM files found in this directory and recognises from then on all the font names stored there.

If working with POSTSCR.TBL instead of the "historical" version without AFM ***GESS* tabs** assumes that the standard character set IBM850 is being used in order to ensure compatibility to the available scripts which are to be interpreted as they are. If the "more modern" version with AFM is being used then the LATIN1 (Windows character set 1252 Western Europe) is valid as standard.

If in an installation the AFM files are used, i.e. if the environment variable GAFM is set then the #define AFM is automatically valid for the ***GESS* Compiler**. Thus e.g.:

```
#ifdef AFM
ADOBELATIN1;
#end
```

can be in the format file. Then whenever the AFM files are used the standard allocation for the LATIN1 character set is inserted. Alternatively it makes sense to import the relevant allocation. The standard file ADOBENAMES.INC contains the same allocation that ADOBELATIN1 carries out. This file may be a good starting point if an individual allocation is required, e.g. for an Eastern European language. Then the following could replace the example above:

```
#ifdef AFM
INCLUDE = MyAdobenames.inc;
#end
```

Incidentally it is recommended to include explicit information about the character set used in all scripts, i.e. either ENCODING = IMB850; or ENCODING = LATIN1; Then they will always be interpreted correctly regardless of the standard encoding by ***GESS* tabs** on the basis of a GAFM.

ADOBELATIN1

Syntax:

```
ADOBELATIN1;
```

As there are also many allocations to ANSI or LATIN1 character set in the AFM files which are not occupied there is an easy method in ***GESS*** to achieve this standard allocation; the ADOBELATIN1 statement. This also includes the € symbol.

2009/12:
Expansion of the allocations using ADOBELATIN1:

copyright	©
registered	®
mu	µ
yen	¥
section	§
paragraph	¶

ADOBENAME
Syntax:

```
ADOBENAME <char> = <name>;
```

or

```
ADOBENAME <number> = <name>;
```

This is used in the editor to allocated ADOBE character names which can be used for output. e.g.:

```
ADOBENAME ä = adieresis;
```

If a character cannot be represented in the editor then the ASCII code can be written instead. The often used ADOBENAMES.INC contains exactly the allocations which are carried out by the ADOBELATIN1 statement. The advantage lies obviously in the fact that individual characters can be allocated differently if necessary using their ADOBE names. If for example the lower case ä (adieresis) is not required in a set of tables then the code can be used for a different character, e.g. for the lower case z-caron.

(NON-PS): The FONT statement can be used to define any number of font names for line printers (see FONT statement). All fonts have the information assigned as to what pitch is required. For ***GESS* tabs** only fonts with a rigid pitch are suitable for line printers.

Target Names:
(PS and NON-PS): A font can either be bound to individual LABELS (see VALUELABELS statement), or to CELLELEMENTS regarding content (e.g. a font for MEAN).

(PS only): Postscript fonts can also be bound to formal table elements, e.g. the LABELS on the X-Axis. All the key words that can be used for DRAWBOX can also be used here. If a box does not contain any text (and cannot indeed contain any, e.g. LABELSET X), the font is not used. Where several fonts are valid, e.g. for MEAN in the DATACELL the fonts have priority for formal table elements, i.e. in the above example the font for DATACELL would be used. A font for LABELS X or LABELS Y would also have priority over the font of a VALUELABELS or an inherited font (see INHERITFONT).

Variable names can also be used as a reference:

```
USEFONT <VarName> = <FontName> SIZE <number> ;
```

Then all the existing labels for this variable will have the same font.

A USEFONT command is valid for all following tables until changed.

FONT (NON-PS)
Syntax:

```
FONT <Fontname> CPI <number> = <ESC-String>;
```

e.g.:

```
Font  LetterGothicItalic15  CPI 15 =
```

```
27 "(s1S" { kursiv } 27 "(s0B" { normal } 27 "(s15H" { Pitch 15 }
27 "(s4102T" { LetterGothic } 27 "&l6C" { VMI : 6/48 Zoll je Row };
```

Every font required for a printer must be listed as to how the font is to be switched on. The syntax for steering strings is described under the heading INITPRINTER/EXITPRINTER. Additionally every font is allocated a name and pitch. As ***GESS* tabs** only calculates using keystrokes for line printers only fonts with a rigid pitch can be used. Additionally in a table only fonts can be used with identical character pitch. This is ensured using the UNITS command.

INHERITFONT (PS)

Syntax:

```
INHERITFONT [ X | Y ] = [ YES | NO ];
```

Using INHERITFONT fonts can be inherited on to the DATACELLS per column or row from the LABELS. INHERITFONT X inherits fonts from the stub, the X-Axis, i.e. per column. INHERITFONT Y inherits the fonts per row from the LABELS from the banner or the Y-Axis. INHERITFONT is ignored if a content font for a number is defined, e.g. USEFONT MEAN for MEAN. The USEFONT defined has priority over the inherited font.

EPS (PS)

Syntax:

```
EPS [ REPLACE | FOREGROUND ] = <FileName> <xPoints> <yPoints> [ [
WIDTH | HEIGHT ] <Points> ] ;
```

EPS (Encapsulated Postscript) provides the possibility of including pictures, logos, graphics etc. in tables. EPS is used to define which EPS file is to be read and where it is to be on the page. xPoints and yPoints are the coordinates in typographical points for the bottom left corner (1/72 inches).

If the height and width is defined the contents of the EPS file is scaled appropriately otherwise the graphic is included in its original size.

The graphic is included in all following tables until the next EPS statement.

Any number of EPS statements can be included. If the key word FOREGROUND is added before the allocation symbol then the relevant file is in the foreground and can therefore overlap parts of the table. Usually the EPS file is in the background.

Using the key word REPLACE the EPS file is not included in the current list and a new list is created instead.

FORMAT

Defines a format for the representation of a particular cell content. If for example a mean is to be a scale with an algebraic sign, a comma as decimal separator and two decimal places then the following would be written (formats should always be written in quotation marks (")):

```
FORMAT MEAN = "+#,##";
```

FORMAT recognises the following control characters:

"."	full stop as decimal separator
","	comma as decimal separator
"+"	always print algebraic sign (+/-)
"#"	wild card character for a number

All other characters in the format string are adopted for output. If the absolute figures for example are to be put in parenthesis then the following statement can be used:

```
FORMAT ABSOLUTE = "(#)";
```

If a dollar sign and a space is to appear before each cumulative value in a table for example then the `FORMAT` statement would be as follows:

```
FORMAT SUM = "$ #";
```

If a `FORMAT` is defined for a percentage (`ROWPERCENT`, `COLUMNPERCENT`, `TOTALPERCENT`) this definition has priority over designation of the number of decimal places made by `DECIMALPERCENT`.

Format for large numbers:
Large numbers are sometimes somewhat difficult to read and symbols to break them down can be very helpful. ***GESS* tabs** recognises a special variation of the `FORMAT` statement for this; the ^ symbol.
Example:

```
FORMAT SUM = "^.3###,###,###,###";
```

The first symbol after the ^ defines the decimal separator to be used, the following number the number of decimal places. The number string left of the decimal separator is divided according to the position of the hash (#). The number 123456.78 would thus be printed `123,456.780` according to the example above.

This remains valid for all tables until changed.

XI Miscellaneous Commands

ENCODING

As ***GESS* tabs** was born as a DOS program and some clients hate nothing more than a change in standard settings, the Char-Set-Encoding from DOS, i.e. IBM850 for North/Middle Europe is set as standard.

This can be changed in two ways: the encoding can be explicitly defined using the ENCODING statement presented here. The preset is then also changed from IBM850 to LATIN1 for all the file types mentioned below if the font meter from AFM files is read instead of POSTSCR.TBL, e.g. controlled by the environment variable GAFM. Using:

```
ENCODING = LATIN1;
```

the script is read in LATIN1 code.

There is a more general form of the ENCODING statement:

```
ENCODING <filetype> = [ LATIN1 | IBM850 ];
filetype = [ DATAFILE | OPENQFILE | EXCELOUT | HG | EXPORTFILE ]
```

Script, DATAFILE and OPENQFILE e.g. can indeed have different coding even during one program.

The script encoding can also be altered to UNICODE (utf16). This happens automatically at the program start if ***GESS* tabs** recognises from the BOM (Byte Order Mark) whether it is a "little endian" or "big endian" UNICODE; in this case IBM850 or LATIN1 is ignored as script encoding. UNICODE files without BOM lead to the program termination. If the TAB file is UNICODE all the include files must be UNICODE as well.

The remaining text input (e.g. DATAFILE, OPENQFILE) is then also interpreted as 8-Bit-Code.

RECODETASKS

The effect of RECODE statements can be restricted to particular task types. Using:

```
RECODETASKS = tabtask;
```

recodes are only carried out by ***GESS* tabs,** and all RECODE statements from ***GESS*** input or CATI etc. are ignored.

COLUMNOFFSET

Syntax:

```
COLUMNOFFSET = <number> ;
```

In ASCII and COLBIN files data is usually fixed in columns and is allocated to a start column using SINGLEQ, MULTIQ or related statements either explicitly or using * implicitly. The explicit allocation using a number can be modified using COLUMNOFFSET. The actual value of COLUMNOFFSET (preset: zero) is added to the start column.

IGNOREDOUBLECASENO

Syntax:

```
IGNOREDOUBLECASENO = [ YES | NO ];
```

***GESS* data entry** normally prohibits duplicated case numbers. This option switches this test off.

IGNORESETFILTER
Syntax:

```
IGNORESETFILTER = [ YES | NO ];
```

Sometimes it makes sense to ignore the `SETFILTER` in the script. The statement:

```
IGNORESETFILTER = YES;
```

causes the script to be carried out as if no `SETFILTER/ENDFILTER` command were present.

IGNOREMISSING
Syntax:

```
IGNOREMISSING = [ YES | NO ];
```

Preset: `IGNOREMISSING = NO;`

`MISSING` values take part in all operations which means that the following:

```
SETMISSING = -1;
...
...
IF var003 LT 2 THEN ....
```

is also `TRUE` if `var003` in this example has the value -1. The processing of data with missing measured values really demands a threeway logic: `TRUE - FALSE - UNDECIDABLE`. The general convention from ***GESS*** at this point distinguishes between the value of a variable which has been drawn to the test and the characteristic that this value can be `MISSING`. And as the global missing value -1 in this case is less than 2 then the test in the example is `TRUE`.

If `IGNOREMISSING = YES` is set then the behaviour is modified globally at this point. The numerical value of variables with the characteristic `MISSING` are then only drawn to the tests `EQ` and `NE`. The other tests `LT`, `GT`, `GE` or `LE` always produce the value `FALSE`.

END;
Necessary indication of the end of the control file.

HIDDENTOVARLIST
Syntax:

```
HIDDENTOVARLIST = [ YES | NO ];
```

This is used to decide whether hidden (system) variables in a variable list with `TO` should also be included.

EXCLUDEFROMTO
Syntax:

```
EXCLUDEFROMTO = { vartype }*n ;
```

Particular variable types which have been formed using `TO` in a variable list can be excluded, e.g. using:

```
EXCLUDEFROMTO = OPEN;
```

the OPEN variables in a list formed with TO would be ignored.

Possible characteristics for variable types:

```
VARIABLE SINGLEQ
MULTIQ
DICHOQ
SCREEN
OPEN
EXECUTE
FILTER
QBLOCK
CODEBLOCK
HIDDEN
RESTRICT
```

IOCHECK

Syntax:

```
IOCHECK = [ ASCIIIN | ASCIIOUT | COLBININ | COLBINOUT ] ;
```

IOCHECK causes the chosen category to be checked for multiple use of columns or punches which would lead to a runtime error. The user is informed of the critical columns etc.

NOIOCHECK

Syntax:

```
NOIOCHECK <varlist> = [ YES | NO ];
```

The variables in the list are excluded from the IOCHECK.

NOCOLCHECK (XGI, XGC etc.)

Syntax

```
NOCOLCHECK = [ YES | NO ];
```

Preset: NO

If using ***GESS*** input or ***GESS*** questionnaire software the allocation of columns is monitored to avoid multiple use. It can however make sense when filtering for example to use identical physical data areas repeatedly. The standard check can be switched off for this.

LISTVARS

Syntax:

```
LISTVARS= <filename> [ options ];
```

option:

```
ASCIIOUT | COLBINOUT | ALL | SPSS | LABELS
```

This produces a list of all the variables known to the system with filtering, title, label and the columnisation in ASCII files in the chosen file.

Usually a list of the variables is delivered as it is read into the ASCII data set (ASCIIIN, DATAFILE or INFILE). All further variables which, e.g. via COMPUTE etc. are produced are usually not included. If the option ASCIIOUT, COLBINOUT or SPSS is defined then only the variables are shown which have been read from the source (SPSS) or have been exported in this format (ASCIIOUT, COLBINOUT).

The additional option LABELS causes the value labels and their codes in ASCIIIN, ASCIIOUT and COLBINOUT to be included in the list. If SPSS is the source this information is produced automatically. In

this case (SPSS) the description is produced in ***GESS* tabs** syntax. Usually this output can very easily be imported into the script using INCLUDE. In this way it is relatively easy to make the corrections to the text so often necessary (shortening the question text, missing hyphens etc.). If the options for SPSS are combined with the option ALL then the variable names as SINGLEQ are also found in the output file. (The others cannot store SPSS).

GESS

Syntax:

```
GESS [ INCLUDE ] <qualifier> = <filename>   [ COLSFROMNAME ]
                                            [ COLUMN <number> ]
                                            [ VARIABLES <varlist> ]
                                            [ CARD <number> ]
                                            [ BITGROUP <number> ]
                                            ;
```

qualifier:= [COLBINOUT]

The ***GESS*** statement produces statements in the file <filename> in ***GESS*** syntax. If the option INCLUDE is chosen the file is automatically read directly after its production as an INCLUDE file.

COLBINOUT has been realised as a qualifier up until now. Here a suitable COLBINOUT statement is generated for every variable in the <varlist> which has been input or called up using XGI or XGC. A coding according to BITGROUP 10 is generated for MULTIQ and DICHOQ. If the VARIABLES option is missing then all variables are processed.

Using COLSFROMVARNAME the starting position in COLBINOUT is derived form the variable names. The variable names then have to follow the pattern:

```
<text><card>_<column>
```

If <card> is missing the card number 1 is used and if <column> is missing there is a syntax error.

Otherwise (if COLSFROMVARNAME is missing) the parameters CARD and COLUMN are evaluated. Starting with the initial column all variables are filed one after another.

SPSS

Syntax:

```
SPSS [ ASCIIOUT ] = <filename>;
```

Produces a SPSS-Command-File which describes all variables which are read from the ASCII-Input in the form of data list, variable labels and value labels. This command can be used directly in SPSS in order to process the data from the DATAFILE in SPSS. The optional specification of ASCIIOUT describes all the variables which are written in an ASCIIOUTFILE.

If SPSSLONGNAMES (see below) is not set the variable names are shortened to eight characters whilst removing blanks and full stops and replacing $ with _ . If variable names become ambiguous due to this change they are replaced by the names GESS_1 - GESS_999. Variable labels and value labels are shortened to 60 characters and the ***GESS*** special characters for hyphenation etc. are removed.

As SPSS does not recognise multi-response variables the individual variables are taken as the basis and carried over to SPSS. FAMILYVARS (MULTIQ) can be changed into dichotomous variables using the SPSSGROUP statement.

REDEFINEVARS are ignored.

SPSSLONGNAMES
Syntax:

```
SPSSLONGNAMES = [ yes | no ];
```

Old versions of SPSS could not use long variable names; ***GESS* tabs** shortened the names where necessary for output in SPSS syntax. In newer versions of SPSS this is no longer a problem as long names can now also be used in SPSS syntax i.e. setting the option to YES.

SPSSGLOBALSEQUENCE
Syntax:

```
SPSSGLOBALSEQUENCE = [ YES | NO ];
```

There are several different opinions among users as to the order in which the variables in an SPSS syntax file should be stored. Classically it was the order in which the variables were defined in the ***GESS*** script. Then I allowed myself to be convinced that clients of clients preferred the physical order of the data in the ASCII data set. This however did not last which is why the classical order can now be used again, see above.

SPSS__
Syntax:

```
SPSS__ = [ YES | NO ];
```

Use of the double underscore when adapting variable names (MULTIQ, DICHOQ).

NOLOGFILES
Syntax:

```
NOLOGFILES = [ YES | NO ];
```

If set to YES the log files gtc.lst, gtc.msg etc are not produced. Then there is also no relevant Viewer Window in gesstabs.exe.

STRICTINPUTCHECK
Syntax:

```
YES or NO
```

Using the preset (YES) numerical input in a multi-columned field is expected to be justified to the right, otherwise there is a numerical error and the variable values are set to BLANKVALUE. If this additional check is switched off the input field is only checked for an interpretable number even if it is e.g. justified to the left in the input field.

Additionally, if set to YES the data rows cannot be shorter than necessary for interpretation of the input. If for example a variable is supposed to be in column 157 and a row is only 156 columns long then the error "line too short for input demand" is produced. If set to NO the variable is set to MISSING.

BLANKVALUE
Normally an input field which only contains blanks is internally set to zero. If these values are however required then the BLANKVALUE command can define a value.
Example:

```
BLANKVALUE = -1;
```

Preset: `BLANKVALUE = 0.0;`

CONTENTFILE
Syntax:

```
CONTENTFILE <option> = <filename>;
```

option:

```
[ TABLETITLE | TOPTEXT | BOTTOMTEXT | VARIABLES X | VARIABLES Y ] [
option ]
```

***GESS* tabs** can produce an ASCII file with a table of contents for a volume of tables. Using the options the parts of the table to be used to describe it are stipulated.

STARTCOLUMN
Syntax:

```
STARTCOLUMN = <number>;
```

The automatic designation of columns using * presumes that there is a previous variable; from this the next free column is calculated, i.e. an explicit column for a variable always has to be designated in every data row. Alternatively a global STARTCOLUMN can be set.

CARDNUMBER
Syntax:

```
CARDNUMBER = startcolumn width;
```

Command that a data set to be read is to be checked for the correct card order. If the column definition of a card number is known then for that position a 1 is always expected on the first card, for the second a 2 etc. Divergence leads to an error log which is shown in the lower error window on screen and where necessary in the LISTFILE.

CASENUMBER
Syntax

```
CASENUMBER = startcolumn width;
```

If the column definition is known for a case number then an identical value is expected at that position for all cards of a case. Divergence leads to an error log which is shown in the lower error window on screen and where necessary in the LISTFILE.

FIXEDPOSITION
Syntax:

```
FIXEDPOSITION <VarList> = [ YES | NO ];
```

Is only evaluated for FAMILY variables. If a FAMILYVAR has the characteristic FIXEDPOSITION then every code is produced at a fixed position in the output: the code 1 in the first column field, the code 2 in the second column field etc. In a certain sense the negative characteristics of FAMILY and GROUP variables are united here.

NB: If e.g. 20 Columns (10 x 2) have been defined for the output of a FAMILY then a label value greater than ten cannot be represented as there is no eleven or further position.

Overall I don't see the point of this but what the client wants, the client gets!

HIDDEN
Syntax:

```
HIDDEN <VarList> = [ YES | NO ];
```

Using the HIDDEN key word the visibility of variables can be controlled when using a system data set via ***GESS* menu**. Normally all the variables defined by the user in the MENU versions of ***GESS* tabs** are visible; internal system variables are not visible. Variables can be "hidden" in the menu by using the HIDDEN commands.
Example:

```
HIDDEN var1 var2 var4 TO var7 = YES;
```

Conversely the system variables which are usually hidden from the user can be made visible in the menu.
Example:

```
HIDDEN SystemCaseNo = NO;
```

LISTFILE
Normally the interpretation of the commands is logged on the screen. This log or parts of it can be directed into a file which is declared as a LISTFILE.
Example:

```
LISTFILE = Tables.Err;
```

If the interpretation is to appear back on the screen as of a certain point this can be achieved using:

```
LISTFILE = con;
```

NOISE
Syntax:

```
NOISE = <value>;
```

NOISE can be used to "add noise" with random figures to all known variables of a data set. <value> defines how many measurement points are to be replaced by random values. value=1 causes a completely random spread. The range of values is defined according to the number of known labels or the upper or lower ranges as they are defined in RANGE. If neither is known the variable values remain unchanged.

This can be used to produce test data sets which conform to the column and label plan (value=1) or the stability of cohesion against measurement errors can be estimated.

All known nuclear variables have noise added if they have not been excluded by the NONOISE statement:

```
NONOISE <varlist> = YES;
```

NORMALIZE
Syntax:

```
NORMALIZE;
```

or

```
NORMALIZE = <varlist>;
```

In the MULTIQ variables the internal storage can look somewhat unconventional after recoding etc., e.g. if a code is repeated. Using NORMALIZE the standard storage is resumed. This is sometimes desirable if data sets produced with COPYFILE are to be passed onto third parties. For the internal processing with ***GESS* tabs** this makes no difference.

IF ... PRINT

To ease the search for inconsistencies in the data or data errors there is a selective print command with which error texts and/or the contents of a variable can be printed.
Syntax:

```
IF <log. Bedingung> PRINT "ErrorText" <Varlist> [ GOTO <varname> ];
```

Example:

```
IF ( Alter LT 17 AND Schulbildung GE 3 )
OR ( Alter LT 6  AND Schulbildung GT 0 )
PRINT  "Unplausibler Ausbildungsgrad" Alter Schulbildung;
```

The messages either appear in the INPUT-DATA-ERROR window on screen or where applicable in the LISTFILE (see above). A summary table of all the errors can be requested using SUMMARY.

During the program run in ***GESS* input** or ***GESS* CAPI** an error window appears. In this case the GOTT components can be used to refuse data entry of the questionnaire run.

Log output in external files:
In the form:

```
IF <log. Bedingung> PRINT PRINTFILE <filename> <varlist> ;
```

in the tabulating mode output can be produced in individually designated files. In this case a title row is produced with the variable names. The variables in the <varlist> must be SINGLEQ (simple VARIABLEs).

IF ... ASSERT
(only for tabulating software gt.exe or gtc.exe)
Syntax:

```
IF <condition1> ASSERT <condition2> [ TITLE "<text>" ] ;
```

If the first condition (condition1) in a data set is true then the comprehensive condition (condition2) must also be fulfilled, thus it can be ensured that for example an interviewee who uses a product has also heard of it or someone with a driving licence is at or above the legal requirement age.

gt.exe and gtc.exe handle this a little differently: gt uses a graphic interface. If the assertion is breached an ASSERT window is opened in which all variables of interest are depicted and the values can be interactively changed. In order to save the changes a COPYFILE should be designated; if this is not the case there is a warning. gtc is without the interface and provides an error log whilst aborting the program run.

LISTON
Syntax:

```
LISTON = NO;
```

Switches the log for the interpretation of commands off completely, LISTON = YES; (preset) switches it back on.

INCLUDE
Defines an INCLUDE file. Commands from the INCLUDE file are interpreted as if they were in place of the INCLUDE commands.
Example:

```
INCLUDE = VARNAME.def;
INCLUDE = Labels.def;
```

This can be used for example to administrate the variable definitions and the VALUELABELS in different files so that changes in the column positions etc only have to be changed in the definition part. In the

same way VALUELABELS and RECODES to fixed variables which have been defined once can easily be repeatedly introduced into the analysis. Any number of INCLUDE files can be used. INCLUDE files can also call up further INCLUDE files (nested INCLUDEs), in the newer versions INCLUDE itself can be nested.

INCLUDE files are first searched for in the current directory and then in the start directory which is where the GT.EXE or XGT.EXE files are found.

ALIGNALPHA
Syntax:

```
ALIGNALPHA = [ LEFT | RIGHT ];
```

Steers the positioning of strings in the ASCIIOUTFILE if an ALPHA variable is produced. Can be justified to the left or the right.

LEADINGZEROS
Syntax:

```
LEADINGZEROS = [ YES | NO ];
```

Preset: NO

Steers the output of numerical values (in the COPYFILE, in the ASCIIOUT or in the DATAFILE of ***GESS* CATI** or ***GESS* data entry** etc). If LeadingZeros=YES the numerical values are filled out with zeros justified to the left. Valid for the whole run.

MAXLINELENGTH

```
<historisch>
```

Defines the maximum length of a row in the input file. Maximum: 50000. Preset: 3000. By designating a lower MAXLINELENGTH storage memory can be saved which can be used for other purposes e.g. for tables. This is particularly relevant if there is a data set in which the cases are made up of many short rows (see CARDS).

In the newer Windows or Linux versions this has no function as the data sets can always be any length.

PRINTEREXIT
Control string which is written at the end of a PRINTFILE. The individual characters are defined in either decimal or ASCII code
Example:

```
PRINTEREXIT = 12 {FormFeed} 10 {LineFeed} 13 {CR};
```

or ASCII codes are mixed with literal strings. Character chains which are to be passed on to the printer unchanged are set in quotation marks.
Example:

```
PrinterExit = 27 "(a4R";
```

27 is the code for the ESCAPE symbol.

Postscript files do not "really" have an initialisation or exit control but the INIT or EXIT control can be used to switch older Postscript printers between Postscript and text modi (e.g. the LaserJet 3 series from HP). In the Postscript mode the control sequence is only produced at the beginning and the end, in text modus at the beginning and end of every page.

PRINTERINIT
Control string which is written at the beginning of the output of a PRINTFILE. See above for coding.

Example:

```
PRINTERINIT = 27 { ESC } 0 { ASCII 0 };
```

SUPPRESSEMPTYTABLE

Usually a table where no cases are relevant is printed as an empty table. Using `SUPPRESSEMPTYTABLE = YES` this page is suppressed.

TABLEMINIMUM

Syntax:

```
TABLEMINIMUM = <number>;
```

`TABLE` tables can be filtered according to a minimum number of cases. In order to suppress an empty table the `SUPPRESSEMPTYTABLE = YES`; must also be set. An adjusting option valid for the tables defined before it.

GLOBALTABLEMINIMUM

Syntax:

```
GLOBALTABLEMINIMUM = <number>;
```

`TABLE` tables can be filtered according to a minimum number of cases. In order to suppress an empty table the `SUPPRESSEMPTYTABLE = YES;` must also be set. Global preset: is valid until the next `GLOBALTABLEMINIMUM` statement.

STOPONFIRSTERROR

```
YES or NO.
```

Preset: `YES`.

If `NO` the input stream continues to be interpreted even if errors occur in as far as the parser can synchronise itself again; possibly the subsequent errors will gain the upper hand. It is recommended to produce a `LISTFILE` in any case in order to log the error and the erroneous input. Stays valid until the next `STOPONFIRSTERROR` command.

The Windows or Linux versions always end the run using an exception.

CSVEXPORT

Syntax:

```
CSVEXPORT = [ <filename> | "" ];
```

New implementation of the good old output of tables in `CSV`-Format (`HG= …`). All text components of the PS-Table are carried over.

EXPORTFILE

Syntax:

```
EXPORTFILE = [ <filename> | "" ];
```

Internal format specific to ***GESS* tabs** or infratest-dimap.
The output files can be marked according to the time, the symbols "`%T`" are expanded to present the time in the file name.

HTML

Syntax:

```
HTML = [ <filename> | "" ];
```

A HTML version of the relevant tables is stored in the file <filename>.html. Additionally a file called <filename>_frames.html is produced. If this is represented in a browser the browser interface is split vertically down the middle and in the left window there is a link list as table of contents.

TABLEFORMATs for HTML output

```
HTMLDOCUMENT
HTMLHEADER
HTMLFOOTER
```

With these TABLEFORMATs the relevant information can be fed into the HTML output.

STYLEFILE

Syntax:

```
STYLEFILE = <filename>;
```

Using the STYLEFILE individual CSS styles can be included. The contents of <filename> are included in the header of the HTML file.

CONTENTKEY

Syntax:

```
CONTENTKEY = [ <text> | <VARIABLE> ];
```

An option following the TABLE statement. If a CONTENTKEY is defined for a table this text is used to write the table of contents in the HTML frame. Otherwise Tab.1, Tab.2, etc. is written. If the text has a valid variable name the VARTITLE of this variable is included.

AUTOCONTENTKEY

Syntax:

```
AUTOCONTENTKEY = [ VARNAME ] [ YVALID | XVALID | NO ];
```

A CONTENTKEY (see above) is automatically allocated; if YVALID then the first variable in the Y-direction of the table, if XVALID the first variable in the X-direction. Usually the VARTITLE is used if this is present otherwise the variable name is used. If the option VARNAME comes directly after the indicating symbol the variable name is always used (if e.g. the VARTITLE is too long to be meaningfully represented in the table of contents).

HTMLFOREGROUND
HTMLBACKGROUND

The background and foreground colours of tables in HTML can be influenced in the script using the two HTML-specific representation elements:

Example:

```
RGB = YES;
HTMLBACKGROUND TABLE = <red> <green> <blue>;
HTMLBACKGROUND DEFAULTBOX = <red> <green> <blue>;
```

The RGB values are, as is usual in ***GESS***, designated in figure ranges from 0 - 1. (The internal colour management is analogue to Postscript.)

HTMLFOREGROUND influences the colour of the writing.

```
HTMLFOREGROUND DEFAULTBOX = <red> <green> <blue>;
```

The foreground and background colours for other parts of the tables are also carried over to HTML tables (e.g. DATABOX, LABELS X etc.).

Generally the BACKGROUND or FOREGROUND values are adopted from the Postscript presentation if there are no HTML-specific colours defined. A special role is played by the background colour of the whole table which is set by either BACKGROUND TABLE or HTMLBACKGROUND TABLE: if this colour is explicitly set the presentation of the table is changed and the HTML borders around the individual table cells usually used are not printed.

INDEXSTYLEFILE

Syntax:

```
INDEXSTYEFILE = <name>;
```

With this a CSS-Style-File can be designated for the index column of the HTML table output.

INSTANTEXCEL

Syntax:

```
INSTANTEXCEL = [ YES | NO ];
```

During the TABLE run an instance of Excel is opened and all the cell contents are carried over. The distribution of the tables onto individual Excel sheets can be controlled by CHAPTERTITLE.

OPENOFFICEDEVIATION

Syntax:

```
OPENOFFICEDEVIATION = YES;
```

The diversion of output from instantexcel into an Open Office Spreadsheet can be achieved. The range of functions for presentation is limited in comparison to Excel; but if Excel is not available the tables can be imported into spreadsheets. Graphics are not produced.

EXCELHIDEUPDATE

If this option is set to YES the Excel interface is only showed by INSTANTEXCEL=YES if a table is finished. This can reduce the processing time for the transfer to Excel.

To control the appearance of tables using INSTANTEXCEL: the following TABLEFORMATs are available:

EXCELNOWRAPTEXT	causes the texts in the text boxes (TOPTEXT, BOTTOMTEXT) to be adopted with the same formatting (row breaks) as the text has in the script.
EXCELNOFONT	only the naked texts or values are carried over to Excel without any fonts. Helps performance for large volumes of tables. Also true for OPENOFFICEDEVIATION.
EXCELDOCUMENT	if this TABLEFORMAT is set the DOCUMENT indicators are adopted from the volume of tables by Excel.
EXCELNUMBERFORMAT	if this TABLEFORMAT is set the numbers with decimal places are explicitly formatted according to the number of decimal places defined which overrides the standard Excel format of dropping zeros as decimal places.
EXCELFRAMES	borders around the Excel table

EXCELCOLOR	adoption of COLOR FOREGROUND or BACKGROUND
EXCELALIGNH	adoption of horizontal cell alignment
EXCELALIGNV	adoption of vertical cell alignment
EXCELPAGEBREAK	generates a changeover at the end of the table
EXCELHEADER	adoption of a HEADER by Excel
EXCELFOOTER	adoption of a FOOTER by Excel
EXCELONELINELABEL	the labels from the banner are adopted by Excel as cells each having more than one row. This can look more attractive and also results in the data cells of more than one CELLELEMENT being contained in one cell (that has to go together somehow). If there are several CELLELEMENTS in one cell then calculation in Excel is no longer possible and graphics cannot be produced.

EXCELFILENAME

Syntax:

```
EXCELFILENAME = <filename>;
```

If EXCELFILENAME is set then output produced in Excel is stored under this file name, Excel is closed and the ***GESS* tabs** run is ended.

EXCELRANGEDELIM

Syntax:

```
EXCELRANGEDELIM = <char>;
```

The transfer to EXCEL via the OLE port is realised as a transfer of strings. In order to define the data source of graphics a complex number of "ranges" may have to be transferred. Unfortunately the EXCEL programs cannot agree which separator should be used to divide the individual areas. "Usually" it is the semi colon and thus ***GESS* tabs** usually conforms to this at the port. Some Excel however prefer to see a comma here so the separator can be designated like so:

```
EXCELRANGEDELIM = ',';
```

EXCELPICTURE

This key word is used to transfer an illustration (PNG-FILE or JPG-FILE) to an Excel table. This then appears above the table.

CHAPTERTITLE (EXCEL)

Syntax:

```
CHAPTERTITLE = <name>;
```

As these strings are passed onto Excel they must fulfil certain criteria: they cannot be too long and they cannot have been used before.

GRAPHTYPE (EXCEL)

If a GRAPHTYPE has been defined for a table a graphic sheet is produced in Excel and the table is transferred in a graphic. The scaling of the value axis can be fixed using EXCELMINMAX. A list of graph types is presented below.

EXCELGRAPHSHEETNAME
Syntax:

```
EXCELGRAPHSHEETNAME = <name>;
```

If this is not defined the sheet names for the graphic bars are generated from the table bars currently active. Names can however also be explicitly given.

EXCELAXISMINMAX
Syntax:

```
EXCELAXISMINMAX = <minvalue> maxvalue> ;
```

EXCELCHARTINVERT
Syntax:

```
EXCELCHARTINVERT = [ YES | NO ];
```

Swaps the allocation of the data on the axes.

EXCELOUT (out-dated)
(In many cases INSTANTEXCEL should be more practical)
Syntax:

```
EXCELOUT = <filename>;
```

EXCELOUT is used to tell the program system to produce a special ***GESS*** specific output format which is interpreted by an import filter in Microsoft EXCEL written by Visual Basic. This import filter produces a page in Excel for every TABLE table and if so desired after each table page a graphic page with an EXCEL graphic for each table. The type and appearance of the graphics can be influenced by the key words TEMPLATE and GRAPHTYPE. EXCELOUT only affects TABLE tables (not e.g. CODEBOOK, PROFILE etc).

TEMPLATE (EXCEL)
Syntax:

```
TEMPLATE = <templatename>;
```

In EXCEL graphical components can be managed in TEMPLATEs which are given a name. These template names can be transferred to ***GESS***; then the ***GESS*** import filter tries to find these templates and uses them to build the graphic. A template is used for all the following TABLEs until a new TEMPLATE name is given.

GRAPHTYPE (EXCEL)
Syntax:

```
GRAPHTYPE = <xlGraphname>;
```

The following graphic types can be requested:

```
xlColumnClustered
xl3DColumnClustered
xlColumnStacked
xl3DColumnStacked
```

xlColumnStacked100
xl3DColumnStacked100
xl3DColumn
xlBarClustered
xl3DBarClustered
xlBarStacked
xl3DBarStacked
xlBarStacked100
xl3DBarStacked100
xlLine
xlLineMarkers
xlLineStacked
xlLineMarkersStacked
xlLineStacked100
xlLIneMarkersStacked100
xl3DLine
xlPie
xlPieExploded
xl3Dpie
xl3DPieExploded
xlPieOfPie
xlBarOfPie
xlXYScatter
xlXYScatterSmooth
xlXYScatterSmoothNoMarkers
xlXYScatterLines
xlXYScatterLinesNoMarkers
xlBubble
xlBubble3DEffect
xlArea
xl3DArea
xlAreaStacked
xl3DAreaStacked
xlAreaStacked100
xl3DAreaStacked100
xlDoughnut
xlDoughnutExploded
xlRadar
xlRadarMarkers
xlRadarFilled
xlSurface
xlSurfaceTopView
xlSurfaceWireframe
xlSurfaceTopViewWireframe
xlStockHLC
xlStockVHLC
xlStockOHLC
xlStockVOHLC
xlCylinderColClustered
xlCylinderBarClustered
xlCylinderColStacked
xlCylinderBarStacked
xlCylinderColStacked100
xlCylinderBarStacked100
xlCylinderCol
xlConeColClustered
xlConeBarClustered
xlConeColStacked

```
xlConeBarStacked
xlConeColStacked100
xlConeBarStacked100
xlConeCol
xlPyramidColClustered
xlPyramidBarClustered
xlPyramidColStacked
xlPyramidBarStacked
xlPyramidColStacked100
xlPyramidBarStacked100
xlPyramidCol
```

These are Microsoft internal names for the graphic types that the graphic engine from EXCEL makes available.

If a graphic name is not found then a DEFAULT graphic is produced. If an empty string is designated as GRAPHTYPE (GRAPHTYPE = "";) there will be no graphic sheet produced after the table sheet.

HG (out-dated)
(is also carried out in Postscript output)
Syntax:

```
HG = [ <HGFileName> | "" ];
```

Defines a file name for the transfer of data in CSV format. The name reminds us of the first application for which it was used, namely Harvard Graphics. The data from all tables defined thereafter are prepared for a "structured ASCII import" to HG (in English HG describes this import as CSV = "comma separated"). With HG = ""; the output is switched off for the following tables.

HGDECIMALCHAR or EXCELDECIMALCHAR
serves to allocate a decimal sign for the output of numerical values with decimal places. Older versions of HG expect a full stop according to American convention. Newer versions of HG (e.g. 2.3) refer to the version of the system software; a German configured version of MS-DOS would expect to use a comma.
Preset: HGDECIMALCHAR = ",";

The last setting is valid for the whole run.

HGDELIMCHAR or **EXCELDELIMCHAR**
The individual elements of a data row are separated from each other with a delimiter symbol. In the preset this is either a comma or a semi-colon depending on the HG version. These values can however be influenced in the HG IMPORT Menu. The last setting made is valid for the whole run.

HGINVERSE
Preset: HGINVERSE = NO;

The data rows for all tables are transferred to HG in the same form as they are in the table, apart from with COMPARE. COMPARE tables are the exception. In order to organise the values in a STACKED BAR the data matrix in the standard case is inverted before the transfer to HG.

If the data of a COMPARE table is to shown in several PIES for example then the standard inverted data matrix has to be inverted again: HGINVERSE = YES; This command is without meaning for all tables apart from COMPARE. The setting made last is valid for all following COMPARE tables until it is changed.

ASCIIOUTDECIMALCAR
Defines CHAR value which is to be used as a decimal separator in ASCIIOUT.

XII Specialities for Advanced Users

Special Characters in ASCII Data Input

The exclamation mark (!) and the dollar sign ($) have a special meaning if found in the first column of input rows.

An exclamation mark is interpreted as the introduction to a comment row. Comment rows are carried over to the COPYFILE where necessary but are not evaluated. During the transfer to the COPYFILE it can happen that the position of the rows is altered slightly: if a case is made up of more than one row and the comment is within one of these rows the comment is pushed to before the first row of the relevant case.

Dollar signs are interpreted as a commanding escape character; the contents of the row are carried over to the parser for the command language. At the moment only the use of the VARNAME and the RECODE command is permitted.

variable redefinition

REDEFINEVARS
YES or NO. Preset: NO. If REDEFINEVARS is set to YES all command rows in the INFILE appear which redefine the input definition of variables already defined. Command rows in the INFILE are identified using a dollar sign ($) in the first column of a row in the INFILE. Commands conforming to the syntax of the VARNAME or RECODE commands are permitted.

If for example variables with the same content are found in two waves of a survey but are in different positions then the information can be evaluated as follows: two INFILES are defined and at the beginning of each INFILE there is the column definition valid for each file. If for example age is coded as of column 27 in the first wave and as of column 36 in the second wave then the following command is defined:

```
Infile = welle1.dat;
InFile = welle2.dat;
Varname = alter 27 2;
```

If not only the physical column structure has changed but also the coding is different then a specific RECODE command for each file can be defined at the beginning of each INFILE.

At the beginning of Welle1.dat is the following row:

```
$ VarName = alter 27 2;
```

and at the beginning of Welle2.dat is:

```
$ VarName = alter 36 2; Recode Alter 6:9 = 5;
```

The explicit command in Welle1.dat is redundant regarding content but serves to clearly document that variables are redefined during the evaluation of these wave files. It is then also necessary if ***GESS* tabs** is to calculate weighting from marginal distributions: as for this all input files have to be read twice there must be a correct column definition for the changeover from the first read of the last files to the second read of the first file as otherwise the column allocation from the last file would still be used.

The use of $VARNAME syntax is not restricted to the beginning of files; in extreme cases several $-rows with redefinitions can be before each individual data row.

RESETREDEFINEVARS

Syntax:

```
RESETREDEFINEVARS = [ YES | NO ] ;
```

Preset : `NO`

If `YES` the preset from the script is reloaded at the beginning of each new file.

XIII *GESS* tabs Command Language

The file with the control commands is interpreted as a format free input stream. The user can format his commands at will to increase readability.

Capital letters are irrelevant for key words and variable names, i.e. `Age`, `AGE` and `age` all refer to the same variable and `COMPUTE` means the same as `compute`. There is one important exception: font names and pre-processor defines are not part of the ***GESS* tabs** language and need to be written exactly as given, with upper and lower case letters etc.

Pre-processor:
***GESS* tabs** recognises several pre-processor statements. The most serve to administrate several variants of a table program in one source using the mechanism of "Conditional Compiling". The pre-processor commands are as follows:

```
#DEFINE
#EXPAND
#UNDEFINE
#MACRO
#ENDMACRO or #MACROEND
#DOMACRO
#DOMACRO2
#IFDEF
#IFNDEF
#ELSE
#END
```

Pre-processor commands are processed before the ***GESS* tabs** program is translated into its internal form. Using `#DEFINE` names are chosen which then are taken as defined; using `#UNDEFINE` they can be deleted. Using `#IFDEF` or `#IFNDEF` ***GESS* tabs** checks whether a name has been defined or not. All ***GESS* tabs** source rows and all `#DEFINE` or `#UNDEFINE` statements between the `#IFDEF` or `#IFNDEF` and the closing `#END` are processed regardless of the logical value of these tests. Using `#ELSE` the alternative rows to be compiled can be determined.
Example:
an `INCLUDE` File called `SETPAPER.INC` contains the following commands:

```
#IFDEF A4
  #IFDEF quer
    PAPER = Height 210 Width 297;
  #ELSE
    PAPER = Height 297 Width 210;
  #END
#END

#IFDEF A5
  #IFDEF quer
    PAPER = Height 148 Width 210;
  #ELSE
    PAPER = Height 210 Width 148;
  #END
#END
```

Using this little program the formats A4 portrait, A4 landscape, A5 portrait and A5 landscape are set. In a ***GESS* tabs** program this can be achieved easily using the following:

```
#DEFINE A4
#DEFINE quer
INCLUDE = SETPAPER.INC;
```

As can be seen from above example pre-processor commands can be nested, the maximum number possible being 20. Pre-processor commands – with the exception of #EXPAND –are as format free as the other ***GESS* tabs** command languages thus allowing the indents in the program to make reading it clearer for the human reader.

The pre-processor commands are also very useful for writing programs which work for postscript and line printers. In order to make this easier only "PS" or "NON-PS" need be set in the PRINTFILE commands. Directly after the PRINTFILE statement the following could be written:

```
#IFDEF PS
  INCLUDE = FMT05.FMT;
#ELSE
  GETPRTSETUP = HP4QUER;
#END
```

Upper and lower case letters are important for DEFINE names: PS and ps are different DEFINES.

#DEFINEs can not only be defined in the source, a DEFINE string can also be given via the command row with the option -D as the parameter:
Example:

```
GTC xyz.TAB -Dascii
```

transfers the string "ascii" to the definition. If the source then contains the following sequence:

```
#IFNDEF ascii
  PRINTFILE PS = xyz.ps;
#ELSE
  PRINTFILE ASCII = xyz.prn;
#END
```

then in contrast to the normal run an ASCII printfile would be produced.

AND combinations between several defines can be relatively easily achieved using nesting. For the evaluation of ORcombinations the squared brackets are used:

```
#IFDEF [ def1 def2 def3 ]
  compute xx = 1;
#ELSE
  compute xx = 2;
#END
```

xx would contain the value 1 if either def1, def2 or def3 is set.

Parallel there is an OR combination for #IFNDEF:

```
#IFNDEF [ def1 def2 def3 ]
  TABLE = aa by bb;
#END
```

The TABLE statement would only be carried out if at least one of the conditions of def1, def2 and def3 is NOT set.

Using #EXPAND place holders can be defined which are replaced during the run.
Example:

```
#EXPAND #kopf var1 var23 mean( var24.num )
...
...
TABLE #kopf BY var17
TABLE #kopf BY var33
```

```
...
```

The name which has been designated using #EXPAND must start with a hash (#). Wherever the designated name appears after that it is replaced by the text which is in the #EXPAND statement up to the end of the row behind the name: in this case by '`var1 var23 mean( var24.num )`'. In this example this is used to produce tables with different heads.

As #DEFINE can be transferred to the command row as a parameter using -D #expand can be transferred by means of -X#<name>:<wert>.

```
#expand #name wert
```

means the same as

```
-X#name:wert
```

as a parameter. If there are blanks in the string to be expanded the whole parameter must be set in quotation marks.
Example:

```
"-X#kopf:variable1 variable2 variable3"
```

Using #MACRO and #ENDMACRO text macros can be defined with parameters. If for example the Macro #TEST is defined as follows:

```
#MACRO #TEST ( &p1 &p2 &p3 )
SUM dummyvar = &p1 &p2 &p3 ;
IF ABS( dummyvar - 100.0 ) > 0.1
PRINT "Summe aus &p1 &p2 &p3 ergibt nicht 100%" &p1 &p2 &p3 ;
#ENDMACRO
```

it can be called up as often as required:

```
#TEST ( f1a f1b f1c )
#TEST ( f2a f2b f2c )
```

etc.

Macros can have up to 50 parameters. The length of the formal parameter names is restricted to 10 symbols. The names of parameter must begin with an ampersand (&). The key word #ENDMACRO means the same as #MACROEND.

The replacements made by a macro are purely text; the order of symbols that conform to the formal parameter names are replaced within the strings or as part of the token. Thus the following is possible:

```
#MACRO #bimbam( &p1 )
  COMPUTE &p1.recoded = &p1;
  RECODE &p1.recoded 1:100=1/101:200=2/201:300=3;
#ENDMACRO
```

#bimbam(Betrag) produces then a variable called "Betrag.recoded".

Macro loops

Using the command #DOMACRO macros can be called up in loops. If there is a macro with just one parameter, e.g. a macro #tab(&1) then it can be repeatedly called up with #DOMACRO.
Syntax

```
#DOMACRO2( <Macroname> <Schleifenliste> )
```

```
#tab( 1 )
#tab( 2 )
#tab( 3 )
```

can be shortened to

```
#DOMACRO( tab 1:3 )

#DOMACRO( tab 1:3 5 6 11:23 )
```
and so on is also possible.

This is not restricted to numerical uses as list constructs can also be used.
Example:
```
#DOMACRO( tab f1 f2 f3 f4,1 f4.2 f4.5 )
```
etc.

#DOMACRO2
Syntax:
```
#DOMACRO2( <Macroname> <Schleifenliste> ; <weitere parameter> )
```

This is an expansion on #DOMACRO. e.g.
```
#DOMACRO2( macroname a b c 1 : 4 ; parm2 parm3 parm4 )
```

After the semi colon there can be (almost) an indefinite number of further parameters, which are transferred as parameter 2 and so on to the relevant macro.

MACROPROTOCOL
Sometimes it is not so easy to find the cause of a syntax error when working with complex macros; only the macro commands can be seen in the source text and not the expanded product. For this reason it is possible to export the expanded macros into a text file where it is easier to check them.
Syntax:
```
MACROPROTOCOL = <filename> [ DOMACRO ] ;
```

All macro call ups are stored as text in this file which can be useful for finding coding errors in macros.

NB: Long macros are expanded in ONE text row. It can be useful to set the editor to word-wrap.

#STARTEXPORT
#ENDEXPORT
SCRIPTEXPORTFILE
These can be used to define parts of the script as a "foreign code" to be exported. If the name of a `SCRIPTEXPORTFILE` is set all parts of the script between `#STARTEXPORT` and `#ENDEXPORT` are carried over into this file. These texts are also processed and modified by the Macro Expander which is the appeal of this construction. Thus it is possible to output variable names produced by nested macros. As ***GESS* tabs** interprets the input without format none of the line feeds are transferred but a vertical line | can be used to force a line feed at that point.

Further pre-processor commands are `#WEIGHT` and `#WEIGHTEND`. They are explained in Chapter XII.

variable names:
Valid variable names can have any length and can be made up of all printable symbols apart from quotation marks if they are set in quotation marks themselves. The first symbol of the variable names must be a letter (A - Z, Ä, Ö, Ü, ß, upper or lower case). If a variable name is to be used without quotation marks (see "Token" below) then all letters, numbers, full stop and underline are allowed. Some other symbols can also be used e.g. "$" but this is not recommended as it is not necessarily supported by later versions of ***GESS* tabs**. Generally it is not permitted to use variables with the same names as system key words or automatically generated system variables.

Examples of valid variable names:
```
alter v_001 frage2 item.1 "Wahlabsicht bei der Bürgerschaftswahl" "Alter+1"
```

Examples for invalid variable names:
```
SINGLEQ TO SORT ROUND 1alter 1000 1.Frage "1.Frage" alter+1 2000.1
```

Comment:
Comments can be strewn in the text at any point and are set in curly brackets. A comment does not count as written; in extreme cases a comment can even be built into a key word or a variable name. Comments in quotation marks are not recognised as comments, i.e. they are part of the string. In other words strings cannot have comments.
Example:
```
compute x = y { hier soll das doppelte berechnet werden } * 2;
```

Additionally there is a comment to the row end: the double oblique // (in the style of C++). The input as of this symbol up until the end of the row is ignored.
Example:
```
// there is nothing relevant in this row
compute x = y * 2;  // this should be doubled
```

The term "Token":
In many cases ***GESS* tabs** expects a "Token" e.g. a variable name or a VALUELABEL. This is either a word like Age or a unit that is built using quotation marks allowing the use of blanks, hyphens etc e.g. "First Vote". This means: 1. LABEL need not be in quotation marks e.g. the LABEL yes. 2. variable names can also contain blanks or other special symbols if they are placed in quotation marks without fail. Both " and ' count as quotation marks. This rule allows the use of ' or " in tokens. Tokens cannot contain line feeds.

Texts:
Texts (VARTEXT, TOPTEXT etc) are also set in quotation marks and differ to tokens in that they are transferred to the relevant part of the tables including vertical spacing and can naturally contain line feeds. Again both ' and " count as quotation marks.

variable lists:
Many commands operate with a list of variables, the VARLIST. This is identified in the relevant command by <Varlist>. A variable list comprises in the simplest case a list of variables.
Example:
```
var1 var2 var3 var4
```

Often it is more economical to use the TO convention. The unambiguous order of the variables is provided by the order in the variable declarations. In order to build a variable list the key word TO can be used to refer to this order and thus to many variables. It is thus simple to write:

```
var1 TO var4
```

Variables are explicitly declared using VARNAME-, SINGLEQ, VARIABLES, COLUMNVARS, VARGROUP, DICHOQ, VARFAMILY, MULTIQ, MAKEGROUP or MAKEFAMILY statements. Variables can also be built using COMPUTE, COUNT or IF statements. If namely the result of the calculation is to be stored in a not yet existent variable this variable is implicitly declared.

Context:
The ***GESS* tabs** language has a context mechanism in connection with the variable lists. If a variable list is empty, i.e. if no variable has been named then ***GESS* tabs** always refers to the last variable produced.

Thus: Whenever <VarList> appears in the syntax definition of a statement this statement can be used in two forms: firstly with a context reference by referring to an empty variable; then the statement refers to the variable defined last. Secondly one or more variables can be referred to explicitly (with a not-empty variable list). Here the TO convention can be used.

Note on the typography of this manual:
In the explanatory text the key words of the command language have been written in a different typeface (`Courier New`) so that the reader can easily recognise them. This need not be adopted by the user, indeed it is often more practical to use upper and lower case letters to emphasise the relevant parts of the key words e.g.: ValueLabels.

Examples are also easy to recognise due to their different format: `Courier New` and indented.

The usual conventions of file names with the extension * and the level of the environment variable ***GESS*** apply.

XIV. For old friends: List of new features

1. COUNT also for VARFAMILYs and VARGROUPs
2. COUNT also with ne, gt, ge, lt, le
3. VARLIST as the result of COMPUTE, IF
4. New: LISTVARS
5. New: SPSS
6. Tabmenu-menu changed: stub/banner
7. Upper/lower case irrelevant for ALPHA-Vars
8. Labelling of frame columns (TOTALTITLE etc) separate for X- and Y-Axis
9. d-Base interface
10. DELETEVARS/KEEPVARS in Systemout
11. FAMILYVARS: Labels inherited on to daughters
12. New: USERAWSFORSTATS
13. New: -D<text> define from command-line
14. New Tableformat: MULTITOTAL
15. Sortclass for VALUELABELS
16. New: HEADER, FOOTER
17. New: PAGENUMBER
18. New: AUTONOANSWER
19. VARGROUP from vargroups
20. COMPUTE COPY
21. IF ... THEN COPY ... ELSE COPY ...;
22. COLBININFILE QUANTUM
23. Difference 0 – 0 rounded off for %
24. New: #EXPAND
25. LISTVARS = <filename>; (documentation of the variable structure)
26. VARGROUPS allowed in the argument list of VARGROUP
27. Nothing new
28. For COMPUTE COPY: Filters are taken into account (lead to MISSING)
29. New: HISTORY = DATABOX <value> <value>
30. ALIGN TABULATOR for Texts, ASCII 9 count as a tab in text boxes
31. New: GLOBALROWMINIMUM, GLOBALCOLMINIMUM
32. Level-info for characteristics or Labels: LEVEL
33. New: CELLSEQUENCE for orders of CELLELEMENTS
34. New tableformat: PERCENTINLABEL
35. New CELLELEMENT: SUMQUOTIENT
36. New: Percent in COLUMNCOUNT (PS). Trigger: TABLEFORMAT PERCENTINLABEL
37. New: PHYSROW and PHYSCOL in FORMAT for HISTORY
38. SIMPLEVAR <name> = groupvar;
39. BCDVAR <name> = groupvar;
40. FRAMEELEMENTS +/-
41. New: CELLELEMENT: DELTAPOINTS SUMQUOTIENT
42. New: BITGROUP <groupvar> = <varname>;
43. PANE: Sorting level
44. Table segments: TOP BOTTOM EXTREME
45. Hyphenated dictionary: SPLITDCTIONARY, DICTMODE, ADDSPLITS, PRINTDICT
46. Colours in Postscript: RGB, BACKGROUND, FOREGROUND, FRAMECOLOR
47. Row spacing: LINEFEEDFACTOR
48. OVERCODE in SINGLEQ, MULTIQ, DICHOQ and VALUELABELS statement
49. COMPUTE LOAD
50. COMPUTE SORT
51. AUTONOANSWER ... LEVEL ...
52. OPENQFILEASSOCFILE
53. SINGLEQ ... OPEN
54. ***GESS*** input/capi (DISPLAY/RANGE/RANDOM/NOMISSING/MINVALUES) (Ch. XIII).

55. IF <value> IN <variable>
56. SORTCLASS OVERCODE <name> ...
57. Table segment: RANGE
58. IF .. PRINT .. GOTO <varname> (capi/input)
59. ALIGNALPHA for ASCIIOUT
60. COPYLABELS/USELABELS
61. LABELS AS and LABELS COPY in SINGLEQ statement etc.
62. COMPRESSCODEBOOK
63. COMPAREVAR mandatory for COMPARE
64. XCOMPARE for COMPARE tables horizontal
65. MEANINCOMPARE (mean in COMPARE/XCOMPARE tables)
66. FRAMECROSS (Boxtype) (intersection for FRAMECELL X and FRAMECELL Y)
67. ABSCOLPERCENT (Abs. and COLUMNPERCENT on one row)
68. MEANTEST (mean and t-Test)
69. DELTASUMPERCENT
70. CONTENTFILE
71. COPYTITLE
72. COPYTEXT
73. LISTVARS [ASCIIOUT]
73a. SPSS [ASCIIOUT]
74. SLICE for SORT-tables
75. AUTOOVERSORT for automatic sorting of overcodes
76. COLOR FOREGROUND or BACKGROUND for shading dependent on result
77. PROFILELINES: explicit format for lines in PROFILE
78. PROFILESCALE: explicit scaling of PROFILE tables.
79. ADDRESSBASE for survey
80. ABANDON
81. SINGLE
82. QUOTA
83. DATE (appointment making)
84. TABLE ADD
85. VARNAMEXINHG, VARNAMEYINHG, TEXTBOXINHG, TABLETITLEINHG
86. DATABOX part in COLOR statement for explicit of columns etc.
89. RESTRICT in XGC, XGI
90. SINGLE as VALUELABEL property
91. PHONE: direct dialling in connection with PHONEBOARD
92. HEADERS/TABULATE: easier tabulation
93. IF ... RECODE
94. RESTRICT for XGI/XGI in connection with ALPHA variables
95. SORT PANE CODE
96. SLICE as a label option
97. CELLELEMENTS as a label option
98. PROJCOLPERCENT: output of projection in the same row as percentages
99. :FORMAT :DESCRIPTION: local format and local description in tables
100. SLICE ... TOP: combination of table slice with cropping of sortings
101. SETFILTER TEXT: Transfer of filter descriptions in VARTEXT
102. TABSELECT TEXT: Transfer of selection description in TOPTEXT or BOTTOMTEXT
103. LSLICE in SORT part of TABLE statement
104. ELASTICITY
105. SPSS [ASCIIOUT]
106. SCREEN: variable masks in CATI/CAPI or Data Entry.
107. LEADINGZEROS
108. MENUFILTER (XGM)
109. MENUMEAN (XGM)
110. MENUHEADER (XGM)
111. FIXEDPOSITION (option for FAMILYVARS)

112. ABSZERODASH (table format: abs. zeroes printed as "dash")
113. VARIATION: Variation coefficients
114. ZRANGE: central area of a variable
115. CONFIDENCERANGE: upper and lower limit of confidence interval
116. PCNTRANGE: upper and lower limit of freely definable percentile areas
117. PERCENTILEDELTA: difference between percentiles
118. PCNTL3: further percentiles
119. PCNTL4: another further percentile
120. MEANCUT: mean ignoring distribution edges
121. TOPCUT: parameter for MEANCUT
122. BOTTOMCUT: parameter for MEANCUT
123. ZVALUE: parameter for ZRANGE
124. MAKEGROUP
125. MAKEFAMILY
126. SPSSGROUP
127. INIT
128. Expanded semantics for IN
129. Expanded semantics for USEWEIGHT
130. TABLEFILTER 1..10: ARRAY OF TABSELECT
131. MAKEFILTER/USEFILTER no longer available
132. RANDOMGROUP for sorting RANDOM orders by block
133. NOISE
134. FILTER <varlist> AS <varname>;
135. SORT: syntax and semantics expanded
136. COLBININCOLS: deviant "card length" in COLBIN
137. REUSE as option for ADDRESSBASE
138. IF ... PRINT PRINTFILE
139. HELPTEXT
140. ALWAYS with LABELS
141. SINGLEQ as new expression for VARIABLE
142. MULTIQ as new expression for FAMILYVAR
143. DICHOQ as new expression for GROUPVAR
144. DATAFILE as new expression for INFILE
145. DUMMYHEAD
146. Constants as parts of TABLE statements
147. SUPPRESSEMPTYTABLE (empty tables not printed if no cases)
148. MARKCELLS with CELLELEMENTS (output of marking as symbol)
149. DRAWBOX ... BOXRADIUS (box with rounded corners)
150. NOCOLCHECK switches off check for column punch for XGI/XGC
151. IOCHECK check for column punch COLBINOUT ASCIIOUT etc.
152. NOIOCHECK switch off IOCHECK for individual variables
153. BITGROUP for code plans in COLBINOUT
154. RESTRICT with variable lists (instead of individual variables)
155. F9 and F10 in Input/Cati/Capi. everything or nothing
156. SYSMISS IN <variable> as general test for "no value"
157. [<number> : <number>] as left argument for IN
158. #MACRO ... #ENDMACRO: more than one row macros mit parameters
159. INDEXVAR as simple variable array
160. COLCHIQU 4-Field Chi-squared-test of all columns in a row against each other (CELLELEMENT)
161. TTEST as new cell element
162. GROUPRECODE for recoding in group variables
163. LABELS MAKE <number>
164. SIMPLEPERCENTILE switches off interpolation with Median/Percentiles
165. PREQUOTA
166. OR combination in #IFDEF/#IFNDEF
176. RESTRICT evaluated on SCREEN

177. EXECUTE also evaluates labels
178. ASSOCFILE, ASSOCVAR, ASSOCEND: combination of several data matrices
179. BOXMINHEIGHT
180. BOXLINEFEED
181. USEFONT for HEADER/FOOTER
182. MINTABLEHEIGHT revised
183. RANDOM = | <varlist> ...
184. EXECUTE ... LASTVAR
185. MIN/MAX : minimum/maximum of several variables in a case
186. GETQUOTA statement
187. Expanded possibility in SCREEN: multi-response variables
188. QBLOCK: remodelling of variable order of for questionnaire rotation etc.
189. TABLEFORMAT LOCALCONTENT
190. ROWCHIQU analogue to COLCHIQU
191. EXECUTE <varname> = ... ;
192. MULTIQ/DICHOQ with NOINPUT
193. Tabs allowed in OPENQFILE
194. Bugfix: CASENUMBER/CARDNUMBER with ASCIIOUT ALL;
195. New table type: TTEST (dependent/independent)
196. TABLEFORMAT SHOWTTMEAN
197. CELLELEMENT/TABLETYPE COLUMNRANGE
198. IN for COUNT
199. @ in texts also on OPEN variables
200. Bugfix: row counter row numbers
201. COMPUTE ADD
202. Data editor XGD as new product
203. ALLSIGNIFICANCE as alternative to LOWSIGNIFICANCE
204. HIGHSIGNIFICANCE as alternative
205. BEEP = YES or NO
206. GESS command
207. IGNOREMISSING
208. IGNORESETFILTER
209. ASCIIOUT default width changed
210. Expanded syntax for IN: IF [2 3 10:12 22] IN .... (not for COUNT)
211. Expanded syntax for IF SysMiss IN for OPEN Vars
212. SEARCHRANGE for ADDRESSBASE
213. PROFILESORT
214. HIDDENTOVARLIST
215. TABLEFORMAT LONGVARTITLE
216. COPYFILTER (inheritance of existent filters)
217. IF <varname> IS <vartype> THEN ...
218. QBLOCK revised <varlist> ... TO ...
219. IGNOREDOUBLECASENO
220. TEXTTABLE
221. ASCIIOUTDECIMALCHAR
222. SORTMEMORY
223. KEYDUMP
224. LISTFILE in ADDRESSBASE
225. COLPERCENTMEAN
226. COLPERCENTSUM
227. COLPERCZ
228. ROWPERCZ
229. COLPERCEQUAL
230. ROWPERCEQUAL
231. IF .... in ABANDON expanded
232. IF .... in DATE expanded

233. ASALPHA as characteristic of OPEN variables
234. FOREGROUND as parameter to EPS statement.
235. TABLEFORMAT GLOBALSORT
236. EXCELOUT
237. TEMPLATE
238. GRAPHTYPE
239. TABLEFORMAT MULTICOLINHG
240. ASSOCFILE BIG ...
241. MINCOLBASE
242. STATIC as characteristic of variables
243. AUTOCLEAR
244. ACCOUNT
245. USEWEIGHT as parameter of ADDRESSBASE
246. ALTXCODE
247. INUSECODE
248. APPOINTCODE
249. COMPUTE SHUFFLE
250. ADDRESSSERVER
251. COLPERCT
252. TESTCOLUMNS
253. CODEBLOCK
254. ENDCODEBLOCK
255. DOCODEBLOCK
256. COLUMNOFFSET
257. PREQUOTA mit DOCODEBLOCK clause
258. Table format SliceLastPage
259. SPSSIN
260. DELIMITEDIN
261. KEY OPENQFILES ...
262. ALPHA keys for ASSOCFILE, OPENQILE
263. INDEXCHARS for column identification in stat. tests
264. COMPUTE ALPHA
265. ENCODING (script, EXCELOUT, EXPORTFILE, HG)
266. RECODETASKS
267. Options SYMBOL, SYMBOLWIDTH for PROFILELINES
268. BLACKLIST as option for ADDRESSBASE
269. ABANDONFILE
270. ASSERTNOANSWER
271. HELPASTEXT
272. DOCODEBLOCK = <name> AS <varname>;
273. ZONEINPUT
274. RESETFEDEFINEVARS
275. Variable names in SPSS data list sorted acc. to columns
276. BLACKLISTCODE
277. TRYCOUNTCODE
278. IF ... ASSERT
279. XTAB statement
280. Adobe AFM files as alternative to postscr.tbl
281. GAFM
282. ADOBELATIN1
283. ADOBENAME
284. SYSTEMFILENO now has labels
285. Expansion of TABLE statement to local table option
286. HG-output also functions for Postscript printers
287. Cell element MODE (modal value)
288. IGNOREMULTIQOVERFLOW

289. Key word SPSSINFILE (not SPSSIN)
290. FIF (filtered IF)
291. FCOMPUTE (filtered COMPUTE)
292. SPSSLONGVARNAMES
293. TESTCOLUMNS = ; for all Columns
294. EXCLUDEFROMTO
295. STARTCOLUMN
296. TEXTWRAP
297. SIGNIF20AND5
298. NOSIGNIFMEANLESS
299. NOSIGNIFMEANGREATER
300. SIGNPERCENTGREATER
301. SIGNPERCENTALWAYS
302. USEFORMATINHG
303. SPSSVARLABTOTEXT
304. MINTEXTHEIGHT
305. COMPUTE SWAP
306. COMPUTE ASCEND
307. COMPUTE DESCEND
308. FILTER clauses in TABLE statement
309. LABELFORMAT
310. NOZEROFILLINLABEL
311. GENERATELABELS
312. EXPORTFILE : %T
313. CSVEXPORT
314. INSTANTEXCEL
315. CHAPTERTITLE
316. EXCELGRAPHSHEETNAME
317. EXCELCHARTINVERT
318. EXCELCHARTMINMAX
319. SPSS_
320. MEAN_PHYS
321. HTML
322. STYLEFILE
323. HTMLBACKGROUND
324. HTMLFOREGROUND
325. CONTENTKEY
326. COLPERCANDSIGN
327. ASSOCFILE DBASEIN
328. ASSOCVAR ALPHA
329. COLPERCENTLINELIMIT
330. ROWCELLMINIMUM
331. DAYOFWEEK
332. COLPERCSTDERR
333. ROWPERCSTDERR
334. TOTALPERCSTDERR
335. COMPUTE ALPHA <var> = "string constants";
336. -X as parameter for #expand
337. AUTOSIGNCHAR
338. AUTOSIGNFORMAT
339. Change in behaviour of COLOR FOREGROUND
340. COLOR FOREGROUND/BACKGROUND for significance tests
341. AUTOCONTENTKEY
342. EXCELFOOTER, EXCELHEADER, EXCELCOLOR etc.
343. SIGNIF20AND10
344. SIGN3LEVELS

345. Expansion of syntax WEIGHTCELLS
346. LOWERCASE for "exotic" significance indices
347. SPSSGLOBALSEQUENCE
348. ENCODING expanded
349. PRINT2LINES2
350. EXCELONELINELABEL
351. EXCELFILENAME
352. PERCENTILEINTERPOL (SIMPLEPERCENTILE removed)
353. New cell element: PHYSCOLPERCENT, PHYSCOLDELTA
354. New cell element: PHYSROWPERCENT, PHYSROWDELTA
355. New cell element: ROWMEANTEST
356. RECODE in LABELS (see VALUELABELS)
357. LABELRECODE
358. COLDEPTTEST
359. MEANCOLDEPT
360. EXCELRANGEDELIM
361. QUANTUMINCHARS
362. FORMAT for large numbers
363. SIGNIFTEXT
364. SIGNIFLEVEL
365. SHOWSIGNIF
366. WEIGHTCELLS AUTOALIGN
367. Read any number of SPSSINFILE
368. Benchmark Tables: COLPCTBENCHMARK, BENCHMARKVALUES, BENCHMARKCOLOR
369. Expanded HTML control: HTMLDOCUMENT, HTMLHEADER and HTMLFOOTER
370. Variables as argument in RECODE, GROUPRECODE
371. MARKCELLS only for COLUMNPERCENT, ROWPERCENT, ABSOLUTE
371. AUTOSIGNCHARALWAYS as addition to AUTOSIGNCHAR
372. EXCELPICTURE for pictures in EXCEL output
373. CELLELEMENTS for TOTALROW
374. EXCEL control: EXCELNOWRAPTEXT, EXCELDOCUMENT, EXCELNUMBERFORMAT
375. New cell element: PHYSCOLCHIQU, PHYSTTEST, PHYSMEANTEST
376. New cell element: COLPCTBENCHMARK
377. GLOBALTABLEMINIMUM, TABLEMINIMUM
378. IGNOREASCOUTDUPL
379. ASCIIOUTFILE DELIMITED ASCIIOUT = <filename>;
380. Unlimited number of SPSSINFILE statements
381. AUTOSIGNCHARALWAYS
382. Variables as arguments in RECODE values
383. SHOWSIGNIF PERCENT = LESS;
384. NOASCIIEXTENSION: switch for ASCII output
385. ALPHA variables for DELIMITEDIN
386. OPENASALPHA: use of verbatims in tables
387. QUALITAB
388. CELLSET: building synthetic CELLELEMENTS for summary tables
389. SLICESTATISTICS: Table of MEAN rows etc. divided per page
390. EXCELHIDEUPDATE
391. MAXCODEBOOKLINES
392. DATEFORMAT
393. COLPERCTMINIMUM
394. LABELSTOTITLE (option for SPSSINFILE)
395. NOINHERITTEXT, NOINHERITTITLE
396. ADOBELATIN1: expansion of standards to © ® µ ¥ § ¶
397. Function expansion: in TOPTEXT and BOTTOMTEXT rows broken where too long
398. NOWRAPINTEXT: or not (see No. 397!)
399. INSTANTPFD: direct production of PDF table volumes.

400. GLOBALTABLEMINIMUM
401. AUTOSIGNIFTEXT
402. Support of scripts in UNICODE
403. INVINDEXVAR: inverse function to INDEXVAR
404. #DOMACRO2 (expanded parameter list in DOMACRO)
405. MACROPROTOCOL (makes expanded macros visible)
406. ADDNAMETOVARTITLE
407. ScriptExportFile, #StartExport and #EndExport
408. CALCULATECOLUMN (column content calculated from column content)
409. Operator precedence rules (at last!!!)
410. MEANROWINDEX and MEANCOLINDEX
411. IGNORETABINTEXT
412. NOLOGFILES
413. SPLITENTRIES: hyphenated dictionary
414. LISTVARS SPSS
415. CONDENSESPSSGROUP (systematic code plans for SPSS automatically concentrated)
416. SPSSGROUPLABELS , SPSSGROUPLABEL1 and SPSSGROUPLABEL0
417. EXCELNOFONT (speeds up transfer to EXCEL)
418. OPENOFFICEDEVIATION (redirects from INSTANTEXCEL → OpenOffice)
419. SPSSSOUTFILE

XIV Appendix

Appendix A: Program Start and Program Run

The programs are called GESStabs.EXE or GTC.EXE.

GTC.EXE
GTC.EXE is the command line version; it is called up in the input command with the name of the script file as the first parameter. Further support files are required: POSTSCR.TBL, LATIN1.ENC and IBM850.ENC.

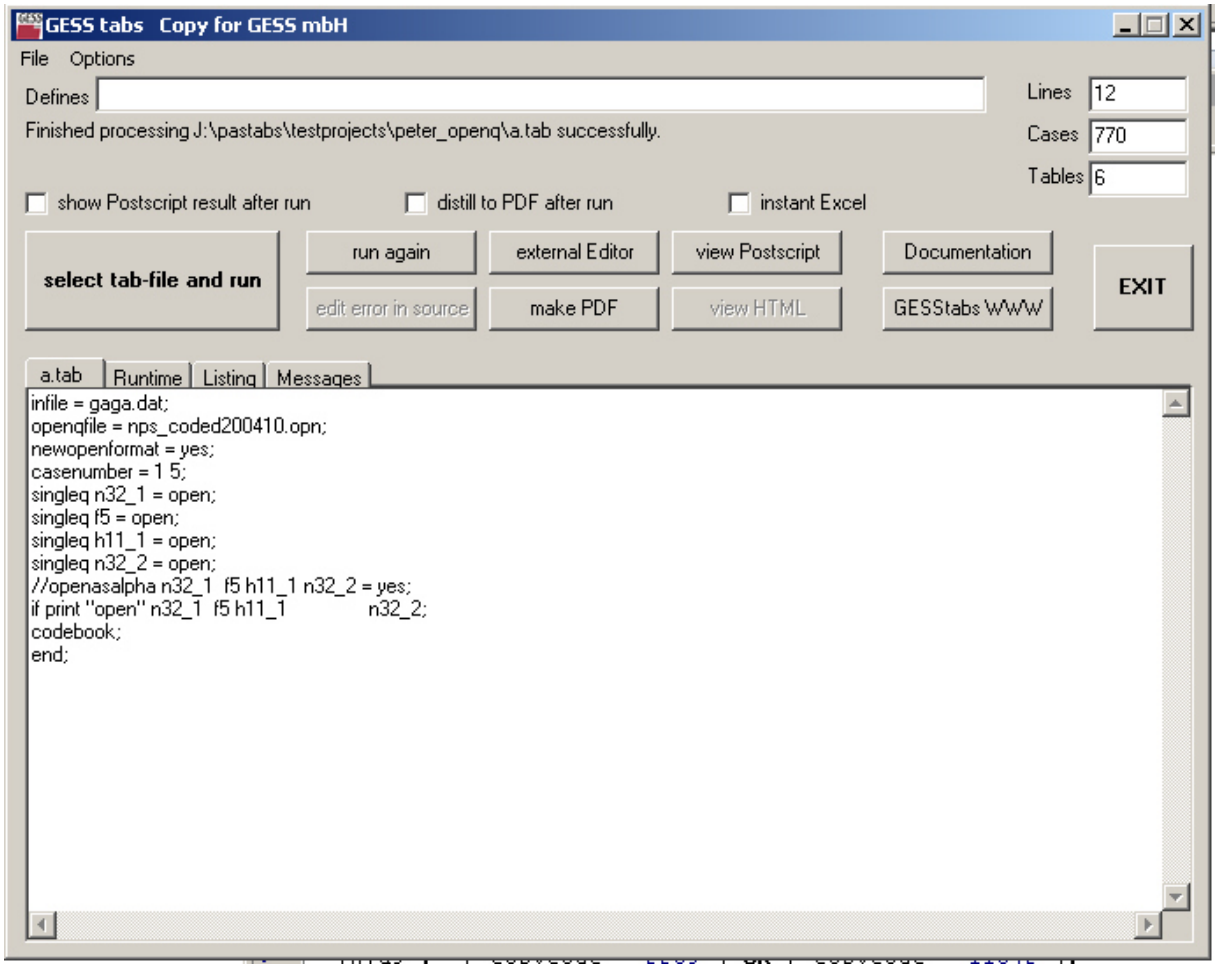
Following further parameter types can be transferred (they always begin with a minus sign):

-D<define>
#DEFINES can be transferred from external sources as -D<define>. -Dtralala has the same effect as the command #define tralala in the script or the text tralala in the ***GESS* tabs** interface define field.

-X<name>=<wert>
#EXPAND as a parameter; "-Xkopf=v1 v2 v3" has the same effect as #expand #kopf v1 v2 v3 in the script. If this parameter contains blanks it must be placed in quotation marks.

-AFM=<afmdirectory>
***GESS* tabs** uses either the old binary metric file POSTSCR.TBL (which is still based on the DOS character set IBM850) or Adobe AFM files. In the latter case the software must be told where the AFM are to be found. There is a standard environment variable GAFM for this purpose which is set as standard for the relevant computer (Windows System Control, SYSTEM, then the tab "expanded". The brave can write it directly in the registry). This information can also be more flexibly transferred using the parameter option -AFM…. -AFM=NO switches off an AFM directory designated by GAFM and forces the use of POSTSCR.TBL.

GESStabs.exe and GTx.exe
GESStabs.exe and GTx.exe together build a slightly interactive version of ***GESS* tabs**. At the heart is GESStabs.exe a start program for an adapted version of GTC.exe. It is simplest to lay both these exe files together in a ***GESS*** directory. GTC.exe naturally also fits in well there. Other support files are also required: POSTSCR.TBL, LATIN1.ENC and IBM850.ENC.



To enable more fluid working the following buttons are available in *GESS* tabs:

Select tab-file and run: a menu for opening files and then starting a tab-file run.

Run again: e.g. the program is rerun after corrections to the script.

External editor: the text file active in the lower window is loaded in to a pre-defined editor.

Edit error in source: is active if ***GESS* tabs** shows a syntax error; in the window the relevant position is highlighted. Clicking in the external editor positions the cursor at the relevant position.

View Postscript: the Postscript File produced with the tables is shown; the Postcript-Viewer can also be configured.

Make PDF: Change Postscript-File to `PDF`.

View HTML: is active if `HTML`-Output is produced. Starts the standard browser.

Documentation: Starts the `PDF`-Viewer and shows the ***GESS*** documentation.

GESStabs WWW: starts the standard browser with the ***GESS* tabs** Website.

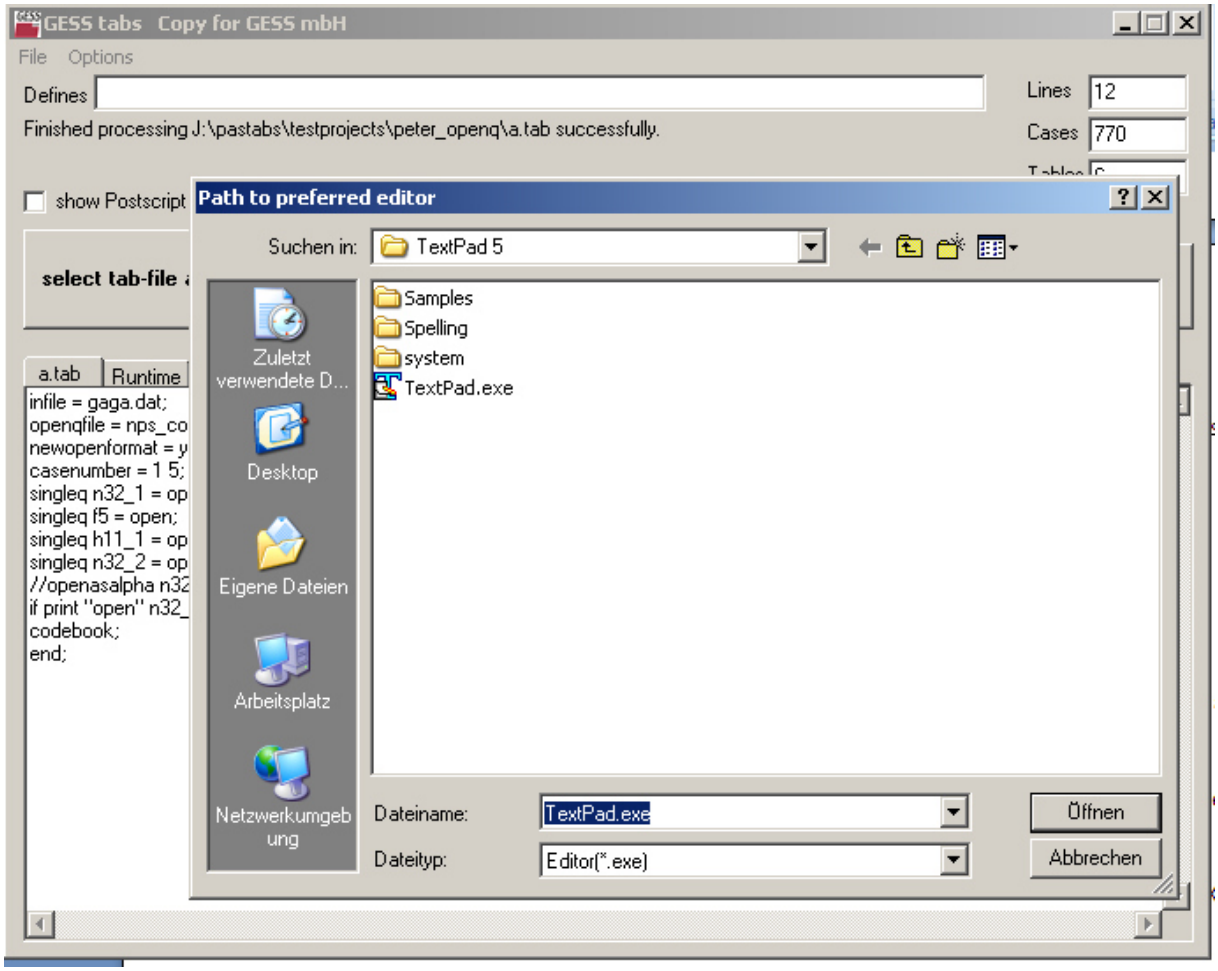
3 further ticks can be made above:

Show Postscript result after run: the viewer is automatically called up after tabulation.

Distil to PDF after run: automatic transference to `PDF` at the end of tabulation. Has the same effect as `INSTANTPDF = YES;` in the script.

Instant Excel: Tables are transferred to `EXCEL` via `OLE`. The same as `INSTANTEXCEL` in the Script.

***GESS* tabs** manages the information which actions are attached to which buttons or check boxes in simple text files with the extension `.txt`. `gtx_exepath.txt` contains the path to the `GTx.exe`, `psviewerpath.txt` to the GS-Viewer (usually `GSView`), `pdfpath.txt` to the PDF-Distiller (usually `Ghostscript`), `editorpath.txt` to the preferred editor (in my case `TextPad`), and `docpath.txt` contains the storage space from `GESStabs.pdf`.



All these files can most easily be produced with the options menu. If the menu point *Set Path to Editor*, is chosen for example then Open File Menu appears and the required program is located (e.g. `c:\Programm\TextPad 5\TextPad.exe`). After the click on "open" the relevant entry is noted in the text file.

At the bottom of the Options Menu there is the menu point *Set Path to AFM files.* If instead of using `POSTSCR.TBL` the metric information from the AFM files is preferred (e.g. in order to use other fonts), the relevant directory can be chosen using this menu point. Any AFM File is "opened" in the directory and then `GTx.exe` recognises all fonts described in this directory. The path information is stored in the file `afmpath.txt`.

Further paths can be chosen using "options": which editor is preferred, which viewer is to be used for Postscript, which software is to be used to distil Postscript to `PDF`, and the path to ***GESS* tabs**

documentation (PDF). If the check boxes for Postscript, PDF, and/or EXCEL are ticked Viewer, Distiller or EXCEL are automatically opened after the run has finished.

The file postscr.tbl must be in the same directory as the EXE files if not using the AFM version of the software.

Appendix B: Index
Words in capitals are key words in the ***GESS* tabs** command language.