8.3 Processing the selected information

You process the data selected by `query` by issuing appropriate analysis instructions.

All paths specified in these instructions are relative to the position last reached by `query` within the database. The system finishes processing all these paths before it starts to process the paths specified in `query`.

8.3.1 The `write` command

The general form of the `write` command is:

PART=<pathlist>	**(default: query[]:total[])**
CONTinue=Yes | No	**(default: Yes)**
MORE=No | Yes	**(default: No)**
SECOnd=Yes | No	**(default: Yes)**
CUMUlate=No | Yes	**(default: No)**
SELF=Yes | No	**(default: Yes)**
POSItion=Yes | No	**(default: Yes)**
STARt=Yes | No	**(default: Yes)**
BRIDge=Yes | No	**(default: Yes)**
IMAGe=Yes | No	**(default: Yes)**
TARGet=Human | Machine	**(default: Human)**

The `write` command is used to print out selected information.

8.3.1.1 The `part=` parameter

The `part=` parameter accepts a list of paths as a value. If it is missing, the system substitutes the path `query[]:total[]`.

8.3.1.2 Controlling the scope of the printout

By default, `write` prints out the maximum possible amount of information extracted from each unit of information addressed by the path, i.e.:

- From each group addressed by `each[]` or `total[]`:
  - the group's name,
  - ordinal number and
  - identifier.
- From each element (every one of which begins on a new line):
  - all entries in all aspects. Every entry begins on a new line.
- From each entry:
  - the core information and
  - the entry's status.
- If the query leads through networks linked by `relation`, the system always issues a message whenever it makes the transition to another group.
- If the data contains information of the `image` data type, this is output to the specified output device. If you issue the `write` command in interactive mode, *κλειω* enables a platform-specific image processing dialogue.

  This dialogue is available only in version 6 of the system. More specifically, its existence creates the difference between version 5 and 6. It is documented in volume A 22 of this series.

You can limit the scope of the printouts described above by invoking the parameters listed below *either* in a `write` command – whereupon they apply to *all* `write` commands in the task being processed – or in an `options` command (cf. section 7.1.6), in which case they will remain in force until you cancel them explicitly or until you terminate the $\kappa\lambda\epsilon\iota\omega$ program.

8.3.1.2.1 The `continue=` parameter

This parameter expects one of the keywords `yes` or `no` as a value. Both of them can be shortened to their respective first character.

- `yes` instructs the system to log every transition to a group using a `relation` information item, hence explicitly restores the system's default setting.
- `no` suppresses this logging function.

8.3.1.2.2 The `more=` parameter

This parameter expects one of the keywords `yes` or `no` as a value. Both of them can be shortened to their respective first character.

- `no` ensures that not more than one entry is printed on each line, and thus restores the system's default setting.
- `yes` instructs the system to join up as many entries as possible on one printed line.

8.3.1.2.3 The `second=` parameter

This parameter expects one of the keywords `yes` or `no` as a value. Both of them can be shortened to their respective first character.

- `yes` instructs the system to print the name of the aspect in front of every secondary aspect ("comment", "original text"), thus explicitly restoring the system's default setting.
- `no` suppresses these identifiers. (So that you can continue to distinguish these aspects from the basic information, they are delimited by the same data signal characters as they were when they were entered – as long as you specify `more=yes` and `cumulate=yes`. Otherwise, the printout displays a series of indentations which do not make a particularly attractive impression.)

8.3.1.2.4 The `cumulate=` parameter

This parameter expects one of the keywords `yes` or `no` as a value. Both of them can be abbreviated to their respective first character.

- `no` instructs the system to begin each element on a new line, thus restoring the system's default setting.
- `yes` instructs the system to combine as many elements as possible on each line.

8.3.1.2.5 The `self=` parameter

This parameter expects one of the keywords `yes` or `no` as a value. Both of them can be abbreviated to their respective first character.

- `yes` instructs the system to print out each element's name, thus explicitly restoring the system's default setting.
- `no` suppresses this output.

8.3.1.2.6 The `position=` parameter

This parameter expects one of the keywords `yes` or `no` as a value. Both of them can be abbreviated to their respective first character.

- `yes` instructs the system to print out the ordinal number and group identifier for every group which is output, thus restoring the system's default setting.
- `no` suppresses this output.

8.3.1.2.7 The `start=` parameter

This parameter expects one of the keywords `yes` or `no` as a value. Both of them can be abbreviated to their respective first character.

- `yes` instructs the system to print out the standard information concerning a group, i.e. the group's name at the very least. Whether any additional information is printed out depends on the value assigned to `position=`. Thus this parameter value explicitly restores the system's default setting.
- `no` suppresses all standard information concerning groups, regardless of the value assigned to `position=`.

8.3.1.2.8 The `image=` parameter

This parameter expects one of the keywords `yes` or `no` as a value. Both of them can be abbreviated to their respective first character.

- `yes` instructs the system to output any `image` information it encounters to the specified output device, thus explicitly restoring the system's default setting.
- `no` suppresses this output.

8.3.1.2.9 The `bridge=` parameter

This parameter expects one of the keywords `yes` or `no` as a value. Both of them can be abbreviated to their respective first character.

- `yes` requests a detailed address to be printed for every entry which points into another database.
- `no` abbreviates this to the name of the database into which the entry points.

8.3.1.3 Dumping contents of a data base into an ASCII file

By default $\kappa\lambda\epsilon\iota\omega$ prepares the output from a `write` command to be read by a human. It can also be used, however, to dump a complete database into an ASCII file as a backup or to transfer the data to another computer system. Furthermore, portions of a database can be dumped into a file to be reorganised according to a different structure.

8.3.1.3.1 The `target=` parameter

This parameter expects one of the two keywords `human` or `machine` as a parameter value. Both of them can be abbreviated to their respective first character.

- `human` asks for output to be formated for a human reader, the default action of the system.
- `machine` asks for output formatted as $\kappa\lambda\epsilon\iota\omega$ input data.

Irrespective of the value of this parameter the output is directed to the current output unit. In almost all cases it will be advisable, therefore, to specify the `target=` parameter of the `continue` or `stop` command whensoever `target=` ist set to `machine` with the scribe command. On these parameters see chapter 3 of this manual.

8.3.1.4 Examples

Below are some examples of correctly written `write` commands:

```
write

write part=family/child:each[]

write part=:name,:date_of_birth,:birthplace

write part=:name,creditor:each[],query[]:age

write part=poem:verse;self=no
```

8.3.2 The `catalogue` command

The general form of the `catalogue` command is:

NAME= <filename>	**(MUST be specified!)**
PART= <list of paths>	**(MUST be specified!)**
LIMIt=<list of characters>	**(default: ”.,;:?!()”’” ”)**
SIGNs=<list of characters>	**(default: ”.?!”)**
NULL=<list of characters>	**(default: none)**
TYPE=Words | Terms	**(default: Words)**
POSItion=Words | Sentences | Part | Element	**(default: none; can be specified more than once)**
OVERwrite=No | Yes	**(default: No)**
FIRSt=Yes | No	**(default: Yes)**
SECOnd=No | Yes	**(default: No)**
LEMMa=Latin	**(default: none)**
SOUNdex=<definition name>	**(default: none)**
SKELeton=<definition name>	**(default: none)**
ALSO=<character>	**(dfault: none)**
SELF=No | Yes	**(default: No)**
FORM=No | Yes	**(default: No)**
REPEat=Sentences | Part | Element | Back | Root	**(default: none)**
MORE=<number>	**(default: zero)**
CLASsification=<definition name>	**(default: none)**
WITHout=Yes | No	**(default: Yes)**

This command is used to set up an index of locations. As a rule, an index of this type contains a list of all words converted to lower case which are contained in the elements specified by the `part=` parameter. $\kappa\lambda\epsilon\iota\omega$ can access any entry on this list instantaneously, without perceptible delay. Lists of this kind are useful for accessing a subset of a database directly, using the `catalogue[]` and `keyword[]` built-in functions (cf. sections 8.1.1.2.2.3 and 8.1.1.2.2.4 of this manual), and also for processing a database interactively, as described in German in “Halbgraue Reihe” volume B10.

8.3.2.1 The `name=` parameter

The `name=` parameter expects a valid filename as a value. $\kappa\lambda\epsilon\iota\omega$ sets up a catalogue under this filename and simultaneously enters this name in the library of logical objects assigned to the currently active database. By default, the system assumes that a catalogue with this name does not yet exist. If one does exist the system will refuse to execute the instruction. You must specify this parameter.

8.3.2.2 The `part=` parameter

The `part=` parameter expects a list of paths as a value. You must specify this parameter.

By default, `catalogue` processes the core information of every entry in the “basic information” aspect of every element addressed by the list of paths.

8.3.2.3 The `limit=` parameter

The `limit=` parameter expects a constant as a value. This parameter defines the characters used to signal the end of a word. Here you should also specify those characters which are not to be included in the catalogue. By specifying the relevant numbers as (sub)values of this parameter you can, for example, instruct $\kappa\lambda\epsilon\iota\omega$ *not* to include numbers in the index of locations.

Regardless of how you define the blank space character in this parameter, the system *always* interprets it as a word separator.

8.3.2.4 The `signs=` parameter

The `signs=` parameter expects a constant as a value. It specifies the characters which are used to mark the end of a "sentence". However, this parameter is only effective if you also specify the parameter `position=sentences`.

8.3.2.5 The `type=` parameter

This parameter expects one of the keywords `words` or `element` as a value. Both of them can be abbreviated to their respective first character. It tells the system what kind of entries to make in the catalogue.

- `words` instructs the system to build up the catalogue from individual words, separated by the separator specified in `limit=`. The maximum word length is an installation option; in the system as delivered, it is set to 40 characters. This is how the system creates catalogues by default.
- `terms` instructs the system to include the contents of entry in question, i.e. including any blank spaces in the entry, as an entry in the catalogue. The maximum number of significant characters to be included is an installation option. In the system as delivered, the limit is set to 40 characters.

8.3.2.6 The `position=` parameter

The `position=` parameter expects one of the keywords `words`, `sentences`, `part` or `element` as a parameter, all of which can all be shortened to their respective first character. You can use it to instruct $\kappa\lambda\epsilon\iota\omega$ to count down to the position in which the respective catalogue entry is held in the original database with the help of a series of scales, while the system is actually setting up the catalogue. Specifying this parameter means that when you are processing the catalogue in interactive mode you can define the entry's context for printing or search operations very much more precisely. Above all, however, you should be aware that if you specify this parameter several times, you may increase the amount of memory space required to store the catalogue by a very large margin.

The system interprets this parameter's values as follows:

- `words` instructs the system to remember exactly to which word in the text specified by the list of paths a given catalogue entry corresponds. The system always interprets the beginning of a new entry as an integral boundary.
- `sentences` instructs the system to remember exactly to which sentence in the text specified by the list of paths a given catalogue entry corresponds. The system always interprets the beginning of a new entry as an integral boundary.
- `part` instructs the system to remember exactly to which entry in a series of entries described by the list of paths a given catalogue entry corresponds.

- `element` instructs the system to remember exactly to which element in a series of elements described by the list of paths a given catalogue entry corresponds. This system always interprets this kind of context by reference to the original database. Thus “in front of” an element, you will always find the identically named element in the database – even if it was not included in the catalogue.

8.3.2.7 The `overwrite=` parameter

This parameter expects one of the two keywords `yes` or `no` as a parameter value. Both of them can be abbreviated to their respective first character. It determines how the system should proceed if a catalogue with the name specified in the `name=` parameter already exists.

- `no` instructs the system not to execute the instruction if this is the case, i.e. to behave as it would by default.
- `yes` instructs the system to delete all parts of the existing catalogue with this name and then proceed to execute the instruction. This means that all the reference lists which depended on this particular catalogue are also lost.

8.3.2.8 Defining access paths

You can set up four access paths in each catalogue:

a) One access path via complete word forms or terms,
b) one access path via the beginnings of words,
c) one access path via word endings and
d) one access path via forms derived from words or terms obtained from the database.

Of these access paths, a) is created implicitly as soon as you set up access paths b) or c). You must set up at least one of access paths b), c) or d).

By default, access path b) – and hence also implicitly access path a) – are both implemented; the system only sets up the other access paths if you explicitly request it to do so.

8.3.2.8.1 The `first=` parameter

This parameter expects as a parameter value one of the two keywords `yes` or `no`. Both of them can be abbreviated to their respective first character.

- `yes` enables the system to access complete word forms and also the beginnings of word forms in the catalogue. This is the system's default setting.
- `no` means that the system cannot access the beginnings of words.

8.3.2.8.2 The `second=` parameter

This parameter expects one of the keywords `yes` or `no` as a parameter value.

- `no` means that the system cannot access word endings. This is the default setting.
- `yes` enables the system to access word forms and also the endings of word forms in the catalogue.

8.3.2.8.3 Access paths via derived forms

Because the system assumes that depending on the type of material of which your data is formed, only one derived form of the corresponding entries should be included in a catalogue, you can only specify *one* of the following parameters for each catalogue.

All procedures used to derive forms are transparent to the user, i.e. you are never confronted by the derived forms. However, whenever you instruct the system to access a given word form, it also obtains for you all word forms from which it is possible to derive the same basic form which can be derived from the word form you specified.

8.3.2.8.3.1 The `lemma=` parameter

Currently, the only keyword this parameter accepts is `latin`. This places at your disposal the lemmatising system of the Istituto Linguistica Computazionale, as developed by Dr. Andrea Bozzi, an earlier version of which had been implemented as a stand-alone program by Giuseppe Cappelli. A more detailed description of this lemmatising procedure is available on request.

Every time the system encounters this parameter with this parameter value, it lemmatises the word forms contained in the catalogue. This means that when you are using the catalogue, you can access the lemmata directly. This way of working with catalogues makes particularly good sense when full-text databases containing Latin texts are involved.

8.3.2.8.3.2 The `soundex=` parameter

This parameter accepts as a parameter value a `soundex` definition, which you must define before the next `continue` command at the latest.

We would advise you to set up this directive with the parameter `type=permanent`, so that you can access it at a later date (cf. section 7.3.1.1.3 in this part of the manual).

If you use this parameter while you are creating catalogues, the system will process the material using the specified `soundex` definition, after which it will make it available in the form of a `soundex` code. This means you can also access the respective code values (cf. section 7.3.1.8 in this part of the manual).

8.3.2.8.3.3 The `skeleton=` parameter

This parameter accepts the valid name of a `skeleton` definition as a parameter value, which you must define the next `continue` command at the latest.

You should always set up the `skeleton` definition with the `type=permanent` parameter, so that the system can access it at a later date (cf. section 7.3.1.1.3 in this part of the manual).

This parameter works in much the same way as the `soundex=` parameter, except that the system processes the material using a `skeleton` definition and then makes it available in this form (cf. section 7.3.1.9 in this part of the manual).

8.3.2.8.3.4 Support for external lemmatization

$\kappa\lambda\epsilon\iota\omega$ can also create catalogues from data which have been entered into the database already in a lemmatized form. In such cases another software system has typically been used during data preparation. The system expects such data to have the form

$$< form >< separator >< lemma >$$

where the *separator* is an arbitrary not reserved character, as in the example

```
Quicunque@quicumque in@in Societate@societas nostra@mio-nuestro,
quam@quam Jesu@nomen-proprium ...
```

The separator may be surrounded by space characters; the lemmata have to consist exclusively of characters which are legal within words, they must therefore not contain any character which has been assigned to the `limit=` or `signs=` parameters. By default every form which is not followed by a separator is not considered for building the access path for derived forms. (It is entered into the primary access path, though.)

8.3.2.8.3.4.1 The `also=` parameter

This parameter expects as parameter value a single character which is used as separator. The use of this parameter and the parameters `soundex=`, `skeleton=` and `lemma=` is mutually exclusive.

The parameter defines which character shall be used as separator between form and lemma.

8.3.2.8.3.4.2 The `self=` parameter

This parameter expects one of the two keywords `no` or `yes` as a parameter value. Both of them can be abbreviated to their respective first character. It decides whether forms without explicit lemma shall be considered their own lemma.

- `no` orders the system explicitly not to consider any form without lemma when constructing the access path for derived forms. This is the system default.
- `yes` orders the system to consider a word without lemma to be itself its own lemma. With this parameter value the following example would therefore be equivalent to the one given above:

```
Quicunque@quicumque in Societate@societas nostra@mio-nuestro,
quam Jesu@nomen-proprium ...
```

8.3.2.9 The `form=` parameter

This parameter expects one of the two keywords `yes` or `no` as a parameter value. Both of them can be abbreviated to their respective first character. It tells the system whether to differentiate between upper-case and lower-case characters in the catalogue. This parameter's value establishes the long-term "sensitivity" of the catalogues: it is *not* possible retrospectively to decrease the sensitivity of a catalogue which has been defined on the basis of this decision. (Obviously it is also not possible to modify a catalogue which you set up to ignore this distinction so that it does start to take it into account .)

- `no` instructs the system to ignore the distinction between upper-case and lower-case characters, i.e. to behave as it would by default.
- `yes` instructs the system to preserve the distinction between upper-case and lower-case characters in the catalogue.

8.3.2.10 The scope of the references included in the catalogue

By default, the system includes a reference to either every word or every term in the processed data in the catalogue. Consequently, if one and the same word appears several times in an entry, this entry is represented several times in the relevant catalogue. Also as a consequence, when you are setting up lists of references interactively (cf. “Halgraue Reihe” volume B10, section 2.3.2.2), you may assume that a list of locations contains as many references to a particular entry as that entry contains terms which qualify for inclusion in your list of locations. So if you set up a list of locations defined as *all contexts containing “a” or “b”*, it will contain an entry in which “a” and “b” both appear, twice over.

You can of course modify the way the system behaves when a criterion appears several times in a defined context.

8.3.2.10.1 The `repeat=` parameter

This parameter expects as a parameter value one of the keywords `sentences`, `part`, `element`, `back` or `root`. All of these can be shortened to their respective first character.

It defines which context(s) in a catalogue, or in a list of references derived from that catalogue, should only appear once.

- `sentences` tells the system that only one instance of a word appearing several times in a single sentence should be included in the catalogue. The system includes the first instance of this word in the sentence in the catalogue. In all calculations with respect to comparisons and display output (cf. “Halbrgraue Reihe” volume B10, section 2.3.2.4), the system ignores any further instances of this word in the sentence. When setting up a list of references, the only reference the system takes into consideration for each sentence is the one which was included as the first in the list.
- `part` tells the system that only one instance of a word appearing several times in each entry should be included in the catalogue, or conversely, that every list of references derived from the catalogue should only contain each entry once. The consequences of using this parameter are similar to those described for `sentences`.
- `element` tells the system that only one instance of a word appearing several times in each element should be included in the catalogue, or conversely, that every list of references derived from the catalogue should only contain each element once. The consequences of using this parameter are similar to those described for `sentences`.
- `back` tells the system that only one instance of a word appearing several times in each group should be included in the catalogue, or conversely, that every list of references derived from the catalogue should only contain each group once. The consequences of using this parameter are similar to those described for `sentences`.
- `root` tells the system that only one instance of a word appearing several times in each document should be included in the catalogue, or conversely, that every list of references derived from the catalogue should only contain each document once. The consequences of using this parameter are similar to those described for `sentences`.

8.3.2.10.2 The `more=` parameter

This parameter expects a number as a parameter value. It allows you to define the meaning of the `back` and `root` parameter values for the `repeat=` parameter more precisely.

If it appears in conjunction with `back`, it specifies the distance of the group which the system should use as the context from the group containing the element which is currently being processed. Hence the combination

```
repeat=back;more=3
```

means that each word in every element at a distance of three logical levels or less from a group containing them all together should only be included once in a catalogue. If the document is less than <number> levels away from the element being processed, the system treats the document as the context.

If this parameter appears in conjunction with `root`, it specifies the distance of the group which should be used as the context from the document containing the element which is currently being processed. Hence the combination

```
repeat=root;more=3
```

means that each word in the group which contains the element currently being processed, and which is three levels away from the document containing this element, should only be included once in the catalogue. If the group directly containing the element is itself less than three levels away from the containing document, the system treats the group as the context.

8.3.2.11 The `null=` parameter

The `null=` parameter expects a constant as a value. It specifies characters which are to be completely ignored during the creation of the catalogue. It is typically used to exclude bracketing symbols, editorial question marks and the like.

8.3.2.12 "Classified" catalogues

A catalogue can be created from elements which contain "classified" text, as described in section 7.3.1.19 of this manual. As a result:

- When the catalogue is used with the full text system, the classifications for the selected term will be displayed.
- Within the full text menus additional submenus can be activated, which allow to restrict text searches to words with a specified combination of classification codes.

8.3.2.12.1 The `classification=` parameter

This parameter accepts the valid name of a `classification` definition as a parameter value, which you must define the next `continue` command at the latest.

You should always set up the `classification` definition with the `type=permanent` parameter, so that the system can access it at a later date (cf. section 7.3.1.1.3 in this part of the manual). Once the definition has been created, the system will not allow its deletion or modification as long as a catalogue exists which uses it.

The catalogue will be created in such a way that for each entry within it a record of its class in all independent subclassifications is kept.

8.3.2.12.2 The `without=` parameter

This parameter expects one of the two keywords `yes` or `no` as a parameter value. Both of them can be abbreviated to their respective first character.

- `no` tells the system to accept every word into the catalogue, even if none of the subclassifications in the applicable classification have been specified. This is the default behaviour of the system.
- `yes` restricts the catalogue to those words for which at least *one* class in one of the subclassifications has been selected. Usually this reduces the size of a catalogue dramatically. For *display* all of the text is still available.

8.3.2.13 Examples

Below are some examples of correctly written `catalogue` instructions:

```
catalogue name=vocabulary;part=letter:text
```

```
catalogue name=vocabulary;part=letter:text;
        overwrite=yes;signs=".?!;";
        position=words;position=sentences
```

```
catalogue name=people;part=:family_name;
        type=terms
catalogue name=vocabulary; part=letter:text;
        first=yes;second=yes;lemma=latin
catalogue name=people;part=:family_name,
        type=terms;first=yes;second=yes;soundex=famname
catalogue name=people;part=:family_name;
        first=yes;second=yes;skeleton=famskeleton
```

8.3.3 The index command

The general form of the `index` command is:

PART=<path>	**(MUST be specified!)**
IDENtification=<path>	**(default: root[0]:sign[])**
POSItion=No | Yes	**(default: No)**
FIRSt=Start | Limit	**(default: Start)**
AFTEr=<definition name>	**(default: none)**
LIMIt=<print constant>	**(default: ” ”)**
SIGNs=<number>	**(default: dependent on data type)**
FORM=Left | Right	**(default: dependent on data type)**
ORDEr=<definition name>	**(default: Order)**
USAGe=<definition name>	**(default: none)**
WRITe=Yes | No	**(default: Yes)**
WITHout=No | Yes	**(default: No)**
PREParation=<definition name>	**(default: none)**
TYPE=Index | Count | Table	**(default: Index)**
SELF=<character string>	**(default: none)**
MAXImum=<number>	**(default: none)**
MINImum=<number>	**(default: none)**
SUBStitution=<character string>	**(default: none)**
CUMUlate=No | Yes	**(default: No)**
MORE=Last | First | Null	**(default: Last)**
REPEat=<character string>	**(default: none)**
CONNect=<comparison filter>	**(default: none)**
SEQUence=<comparison filter>	**(default: none)**
CONTinue=<comparison filter>	**(default: none)**
LINEs=No | Yes | <character>	**(default: No)**
ALSO=<character string>	**(default: none)**
SOURce=<number>	**(default: none)**
ALWAys=<character>	**(default: none)**
ONLY=<character>	**(default: none)**

You can use this command to create sorted lists of characteristics or combinations of characteristics. You can then use these lists in your project, or in order to create indices for publishing. For this reason, we have provided a series of parameters which you can use for the first purpose if you wish, but which are less convincing in that role than they are as tools for creating formally faultless indices.

Unlike most other commands, this command accepts several `part=` parameters. Each of these `part=` parameters – and every `identification=` parameter – defines one *logical column* in the index. Additionly, unless you use `identification=` explicitly to define another location specification or `type=number` to request such a specification for counting purposes –an extra logical column is included in the index, in which the identifer of the document from which the contents of the first logical column were derived, is displayed as a location specification.

The system starts by combining these logical columns into *logical lines* which are then sorted.

The user may use various parameters to ensure that logical columns colliding with logical lines in the index are combined before they are printed out.

The system sorts the index according to the entries requested by the first `part=` parameter. If these are identical for several lines, the system sorts these lines according to the criteria in the second `part=` parameter. If there is still no difference between certain entries, the system calls in additional information until it has been able to sort each entry uniquely.

8.3.3.1 Defining logical columns

You can use the following parameters to define the logical columns into which the index is divided. On every logical line of the index, each logical column contains exactly one entry from the database. If several entries are available for the paths described below, the system constructs one logical line for every possible combination of these entries.

8.3.3.1.1 The `part=` parameter

The `part=` parameter expects a path as a parameter value. The entries in this path are all combined in one logical column in the index.

The system uses the following default settings in order to prepare each separate data type for sorting:

- `text`: the entire contents of every entry are included in the logical column. It is then left-justified and sorted.
- `date`: every entry is sorted in the format *year.month.day*, where the year (with leading zero if necessary) is always four-digit, and the day and month (with leading zeros if necessary) are always two-digit. Calendar dates defined as intervals of time are sorted into exact date specifications according to their *terminus post quem*.
- `number`: the entries – in the standard `number` print format, i.e. to six decimal places – are incorporated in right-justified form into a twelve-digit field. This is a way of ensuring that the system correctly processes small numerical values of the type frequently found in historical sources. If larger numbers occur or uncertainty indicators and/or numerical ranges need to be sorted, you should define a new field width using the `signs=` parameter. If you do so, you should also *explicitly* specify `form=right` (right justification).
- `category`: the system left-justifies these entries, delimits them with commas and sorts them according to the print format defined in the relevant `category` definition.
- `relation`, `location` and `image`: the system converts data items of this data type into their `text`-type representation before inserting them in the index.

The first `part=` parameter in each `index` command has a special significance: the element specified in this parameter *must* exist so that the system can generate an index line. This means that in the case of a database with a widely varied information content, the creation of indices where the element which should appear in the first column only rarely exists, is limited from the start to index lines which are as complete as possible.

There are many occasions, however, on which you will want to include index lines in the index even though the first-mentioned element is missing. If you do want to do this, you should assign the element function `form[]` to the first `part` parameter, with a dummy constant as its argument if necessary (i.e. `:form[""]`). This represents an element specification which – formally speaking – exists in that group, which in turn means that the system can create an index line for every group selected by `query`.

8.3.3.1.2 The `identification=` parameter

This parameter also expects a path as a parameter value. The entries in this path are all combined in one logical column in the index. The only difference between this parameter and the preceding one is that when it comes to defining such logical columns more precisely, a larger number of parameters are available to the system than is the case for columns generated for the `part=` parameter.

The system assumes that the entries specified in this parameter are identifiers suitable for identifying locations, i.e. that as a general rule each entry is not more than twelve characters long.

8.3.3.2 Parameters relating to the index as a whole

Each of the following parameters relates to the appearance of the index as a whole. This means that you can specify them before the first column-defining parameters, and that you only have to do so once.

8.3.3.2.1 The `position=` parameter

The `position=` parameter expects one of the keywords `yes` or `no` as a value.

- `no` instructs the system to output all logical columns for which you did not specify a `signs=` parameter one after the other, separated by a blank space of the character string you assigned to the `limit=` parameter. This is the system's default setting.
- `yes` instructs the system to set up each index line in tabular form. This means that the system attempts to transfer every logical column into a printed column of the same width, beginning at the same print position on every line. The width of these columns is calculated as follows:
  - First, the system reserves thirteen characters for every logical column set up by the `identification=` parameter, or for the system's default identifier.
  - The sum of these column widths is subtracted from the width of the current output line. (defaults: screen output 80 characters per line, printed output 132 characters per line. You can redefine these settings using the `options` command's `signs=` parameter: cf. section 7.1.3.1.2.)
  - The system divides the remainder of the line by the number of logical columns defined in `part=`.
  - Finally, the system subtracts one from this column width, in order to have one print position available as a column separator.

8.3.3.2.2 The `first=` parameter

This parameter expects as a parameter value one of the keywords `start` or `limit`. You can use this parameter to control the order in which the index is sorted.

- `start` explicitly instructs the system to behave as it would be default. The index is sorted in ascending order.
- `limit` instructs the system to sort the index in descending order.

8.3.3.2.3 The `after=` parameter

This parameter expects the name of a `conversion` definition as a parameter value (cf. section 7.3.1.7.)

If you are using complex sets of rules to replace values appearing more than once, or to combine successive lines, or to link together successive but sometimes missing logical columns – in short, if you are using complex index definitions, you may sometimes find that the resultant index lines do not fully live up to your expectations, inasmuch as they contain a construction which results in an inelegant printout, such as lines containing several commas in a row, because they are continuing to display column separators even though there are no values in these columns.

This parameter is designed to help you deal with precisely this situation. The system applies the `conversion` definition you specify to the completed index line, *after* building up the latter in its entirety from the various instructions for the individual logical columns, but immediately *before* printing it out or displaying it.

8.3.3.3 Parameters relating to individual columns

Each of the following parameters helps you to process a single logical column. For this reason, they should only appear after the first `part=` or `identification=` parameter, and each refer to the last parameter of this type to appear.

8.3.3.3.1 Parameters which can be used for any logical column

8.3.3.3.1.1 The `limit=` parameter

This parameter expects a print constant as a parameter value. This constant is output as a piece of text bridging the gap between the logical column for which the parameter is specified and the next logical column. If you specify this parameter for the last logical column in the command, but you did not specify either an `identification=` or a `type=number`, the text is used to introduce the location reference; otherwise it is simply ignored.

8.3.3.3.1.1.1 Print constants

Print constants are a system concept which "has not yet been implemented". In principle, they comprise alphanumeric constants containing sequences which control the output from your printer. In the case of this one parameter the system recognises the following control sequences for the time being:

- `\n` Go to the beginning of the next line.
- `\f` Go to the beginning of the next page.
- `\t` Go to the printer's next tab position.

8.3.3.3.1.2 The `signs=` parameter

This parameter expects a number as a parameter value. This instructs the system to fit the output for this logical column into a printed column with a fixed length of *number*. If an entry is more than *number* characters long, the extra characters are ignored.

If you specify `form=right` *without* specifying `signs=` the system automatically assumes a value for `signs=` of `18`.

8.3.3.3.1.3 The `form=` parameter

This parameter expects one of the keywords `left` or `right` as a value.

- `left` means that the entries are sorted left-justified.
- `right` means that the entries are sorted right-justified and sorted.

8.3.3.3.1.4 The `order=` parameter

This parameter expects as a parameter value the name of a sorting order defined by an `order` definition (cf. section 7.3.1.14).

It instructs the system to sort the logical column specified by the last-encountered `part=` or `identification=` parameter according to the rules of this sorting order.

8.3.3.3.1.5 The `usage=` parameter

This parameter has not yet been implemented.

8.3.3.3.1.6 The `write=` parameter

This parameter expects one of the keywords `yes` or `no` as a parameter value.

- `yes` instructs the system to output the logical column in the index line. This is the system's default setting.
- `no` instructs the system *not* to output the logical column. This means you can sort the index by information which does not then appear in the output.

8.3.3.3.1.7 The `without=` parameter

This parameter expects one of the keywords `yes` or `no` as a parameter value.

- `no` instructs the system not only to output the logical column, but to use it for sorting purposes – according to the position in which it appears in the index line. This is the system's default setting.
- `yes` instructs the system to output the information but *not* to use it for sorting.

8.3.3.3.1.8 The `preparation=` parameter

This parameter expects as a parameter value the name of a `conversion` definition.

It instructs the system to apply this `conversion` definition to the contents of the logical column *before* proceeding to sort the latter. This means that this conversion affects both the order in which the printed lines are sorted, and their appearance, whereas the parameter of the same name in the `order` definition's `conversion` directive relates to the sort operation exclusively.

8.3.3.3.2 Parameters for logical columns generated by `part=`

8.3.3.3.2.1 Parameters affecting single logical columns

8.3.3.3.2.1.1 The `type=` parameter

This parameter expects one of the keywords `index`, `count` or `table` as a parameter value. It is used to define the form of the output.

- `index` instructs the system to generate an index by outputting every line for which entries exist in the logical columns – even if this means that several totally identical lines are printed out one after the other. At the same time, if you have not specified an `identification=` parameter, this parameter instructs the system to output the identification of the document which originated the entry in the first logical column on each index line, by default. This is also the system's default setting.
- `count` instructs the system to print out a single line for each combination of values in the preceding logical columns. This line represents the frequency of these combinations. No identifier is output unless requested by `identification=`.
- `table` has not yet been implemented.

8.3.3.3.2.1.2 The `self=` parameter

This parameter expects an alphanumeric constant as a parameter value. This string is entered in the logical column if no entry exists for it, but the system prepares an index line in any case due to the existence of other logical columns. This parameter only affects the output, not the sort. "Empty" logical columns are *always* sorted before any columns for which information does exist in the database.

8.3.3.3.2.1.3 The `maximum=` parameter

This parameter expects as parameter value a number restricting the number of lines of output. If you did not specify a `type=` – or you specified `type=index` – the system outputs a maximum of *number* lines with the same content for each entry in the relevant logical column; if you specified `type=count`, the only lines printed are ones where the frequency of the entry in the logical column does not exceed *number*.

You can use this parameter to prevent the system from producing a list containing several hundred occurrences of *Smith John* when you really want to concentrate on the "exotic" aspects of your material.

8.3.3.3.2.1.4 The `minimum=` parameter

This parameter also expects as a parameter value a number limiting the number of lines of output. In `type=number`, the system only outputs lines where the frequency of the entry in this logical column does not exceed *number*. You can use this parameter to prevent the system from producing a list of articles of clothing and their associated values comprising several hundred index lines when the ones which really interest you are the ones occurring more frequently, hence more typical of their time.

8.3.3.3.2.1.5 Exclusion of text from sorting

In some applications it is necessary to exclude portions of the text from sorting. For the most typical applications there exists the possibility of preparing the data by a `conversion` directive (cf. section 8.3.3.3.1.8 above) or explicitly marking text to be ignored for sorting purposes (cf. section 7.3.1.14.2.2). Beyond that it can be useful to consider only that part of a “logical column” for sorting purposes, which is is in front of some character (e.g. the first full stop). For such purposes two parameters are available.

8.3.3.3.2.1.5.1 The `always=` parameter

This parameter expects a single character as value. What is in front of this character within a logical column will *always* be used for sorting purposes. The remainder of the logical column will be ignored in sorting, if the full stop is immediately followed by one or more space characters.

8.3.3.3.2.1.5.2 The `only=` parameter

This parameter expects a single character as value. It is meaningful only, if specified together with a `always=` parameter. The character specified with `always=` will signal the end of the text to be considered for sorting only then, if it occurs in front of the character assigned to the `only=` parameter.

8.3.3.3.2.2 Parameters affecting logical columns and lines

The following parameters instruct the system to combine the contents of adjacent logical columns to form a single column of output in certain circumstances, or else to combine a series of logical lines to form a single line of output.

8.3.3.3.2.2.1 The `substitution=` parameter

This parameter expects an alphanumeric constant as a parameter value. If the logical column for which you set this parameter displays the same contents on two or more output lines in succession, these contents are replaced by this character string from the second output line onwards.

8.3.3.3.2.2.2 The `cumulate=` parameter

This parameter expects one of the keywords `yes` or `no` as a parameter value. You should only use it if you used the `cumulate=` or `substitution=` parameters for the previous logical column.

- `no` means that the value of the last-defined `substitution=` parameter relates exclusively to the logical column for which it was specified. This is the default setting.
- `yes` means that the `substitution=` parameter affects each column from the one for which it was set to the last column for which `cumulate=` was specified in unbroken sequence in a cumulative fashion: you can use it, for example, for an index to tell the system that where adjacent logical columns include first and family names, both logical columns should be replaced by a single substitution symbol.

8.3.3.3.2.2.3 The `more=` parameter

This parameter expects one of the keywords `last`, `first` or `null` as a parameter value. It is used to specify which of the constants specified in the `limit=` parameter should be output if a `cumulate=` parameter results in the replacement of several logical columns for which `limit=` was specified.

- `last` instructs the system to output the `limit=` parameter for the last logical column – affected by `cumulate=` – on an index line. (This is the system's default setting.)
- `first` instructs the system to output the `limit=` parameter for the first logical column – affected by `cumulate=` – on an index line.
- `null` instructs the system to ignore all `limit=` parameters for the affected logical columns, and to output a blank space instead.

8.3.3.3.2.2.4 The `repeat=` parameter

This parameter expects a constant as a parameter value.

It instructs the system to combine index lines where the preceding logical columns match to form a single index line. This means that the differing contents of the logical columns for which `repeat=` was defined are separated by the constant specified as the parameter value and then appended to each other, but in such a way that each value appearing in this logical column is only used once.

This parameter remains in force, with the constant you specified, until:

a) the system encounters another `repeat=` parameter,
b) the system encounters a `substitution=` parameter which it treats like an extra `repeat=` parameter after the first `repeat=` parameter,
c) the system encounters a `sequence=` parameter.

Following these rules, the parameter `repeat=" and "` will cause the five index lines below:

```
Huber, Julius:  123-12/4; farmer
Huber, Julius:  123-12/4; weaver
Huber, Julius:  321-9/3; innkeeper
Huber, Julius:  321-9/3; weaver
Huber, Julius:  321-9/3; farmer
```

to turn into two index lines:

```
Huber, Julius:  123-12/4; farmer and weaver
Huber, Julius:  321-9/3; innkeeper and weaver and farmer
```

8.3.3.3.3 Parameters for logical columns defined by `identification=`

Identifications shall frequently be sorted in a more complex way than the alphabetic or numeric one: particularly if they contain shelfmarks like "123-4a" which shall be sorted between "123-5" and "123-4". Additionally it is often desirable to combine lines which are identical except for their identifications into single lines, like merging the two lines

```
Huber, Julius:  123-12/4
Huber, Julius:  123-12/5
```

into the index line:

```
Huber, Julius:  123-12/4 f.
```

This is accomplished by *comparison filters* which serve as parameter value for three parameters (`sequence=`, `connected=` and `continue=`) which provide the services described initially with different degrees of thoroughness. Only *one* of these parameters may be used with a specific `identification=` parameter, as they are mutually exclusive.

Comparison filters comprise a series of format identifiers, i.e. individual characters which specify what kind of count to expect in an identifier, together with separators, i.e. any characters which have not been defined as format identifiers.

The system recognises the following format identifiers:

- `0` Count by Arabic 'page' numbers; 2 appears before `11`.
- `1` Count by Arabic 'section' numbers; 2 appears after `11`.
- `x` Count by Roman numerals.
- `9` Count by Arabic and Roman 'page' numbers; Roman numerals appear before Arabic numbers.
- `a` Count by letters. Each letter is treated as a section identifier: hence `ab` appears before `c`. The distinction between upper and lower case is ignored.
- `A` Count by letters. Each letter is treated as a section identifier: hence `ab` appears before `c`. Upper-case letters are treated as higher-order sections: hence `b` appears before `Aa`.
- `b` Count by letters. Each letter is treated as a number: hence `z` appears before `aa`. The distinction between upper and lower case is ignored.
- `B` Count by letters. Each letter is treated as a number: hence `z` appears before `aa`. Upper-case letters are treated as higher-order sections: hence `b` appears before `Aa`.
- `z` Alphabetical suffix: you can use optional letters to define a preceding numerical element more precisely if required.

If several format identifiers appear in a row (with separators between them if you wish), the system treats them as being in descending order of importance from left to right. If an identifier does not match the comparison filter, the system does not compare it to the preceding one unless it matches the latter precisely. If the system reaches the end of an identifier before it reaches the last format item in the comparison filter, any items which are not present are treated as zero. The system ignores any parts of the identifier which appear after the end of the comparison filter.

Examples:

Counting filter	Applicable to	Not applicable to
`sequence="0-0/0"`	`123-18/4; 123`	`123a-18/4`
`sequence="0z-0a/0"`	`123-18c/4; 123`	`123a-18/4`
`sequence="0z-0z/0"`	`123-18c/4; 123; 123a-18/4`	
`sequence="x 0 a"`	`xvii 12 a; x; x 13`	`24`
`sequence="9"`	`xvii; 23; 125`	`19a`

The system treats matching index lines according to the following rules. In each of the following examples, we use the comparison filter "0-0/0".

8.3.3.3.3.1 The `sequence=` parameter

This parameter is used most oftenly for the purposes indicated at the beginning. Its operation is described by the following rules:

Rule 1: *If two index lines in front of the identifier match, and if both identifiers are also the same, the system only outputs one index line.*

This means that

```
Huber, Julius:  123-12/4
Huber, Julius:  123-12/4
```

turns into the following index line:

```
Huber, Julius:  123-12/4
```

Rule 2: *If two index lines in front of the identifier match, but the two identifiers are not sequential, the system outputs one index line in which the two identifiers are separated by a comma.*

This means that

```
Huber, Julius:  123-12/4
Huber, Julius:  242-3/1
```

turns into the index line:

```
Huber, Julius:  123-12/4, 242-3/1
```

Rule 3: *If two index lines in front of the identifier match and the two identifiers are in sequence, the system outputs one index line with an* `f.` *appearing after the first identifier.*

This means that

```
Huber, Julius:  123-12/4
Huber, Julius:  123-12/5
```

turns into the index line:

```
Huber, Julius:  123-12/4 f.
```

Rule 4: *If several index lines in front of the identifier match and the identifiers are in sequence, the system outputs one index line in which the first identifier is followed by a hyphen, followed in turn by the last identifier in the sequence.*

This means that

```
Huber, Julius:  123-12/4
Huber, Julius:  123-12/5
Huber, Julius:  242-3/1
Huber, Julius:  321-9/3
Huber, Julius:  321-9/4
Huber, Julius:  321-9/5
```

turns into the index lines:

```
Huber, Julius:  123-12/4 f., 242-3/1, 321-9/3 - 321-9/5
Huber, Julius:  321-9/3 - 321-9/5
```

Rule 5: *If several index lines in front of the identifier match but the identifiers do not follow any particular order, the system outputs one index line in which the separate identifiers are linked together by applying the preceding rules, one after the other.*

This means that

```
Huber, Julius:  123-12/4
Huber, Julius:  123-12/5
Huber, Julius:  242-3/1
Huber, Julius:  321-9/3
Huber, Julius:  321-9/4
Huber, Julius:  321-9/5
```

turns into the index line:

```
Huber, Julius:  123-12/4 f., 242-3/1, 321-9/3 - 321-9/5
```

Rule 6: *If several index lines in front of the identifier match and additional logical columns follow after the identifier, these columns are appended to the end of the index line, separated by commas. In addition, the rules described for the* `repeat=` *parameter (cf. section 8.3.3.3.2.2.4) also apply. The separator between each field is a comma, or the value of a* `repeat=` *parameter if you have specified one.*

This means that

```
Huber, Julius:  123-12/4; farmer
Huber, Julius:  123-12/5; weaver
Huber, Julius:  242-3/1; innkeeper
Huber, Julius:  321-9/3; innkeeper
Huber, Julius:  321-9/4; weaver
Huber, Julius:  321-9/5; farmer
```

becomes the index line:

```
Huber, Julius:  123-12/4 f., 242-3/1, 321-9/3 - 321-9/5; farmer,
weaver, innkeeper
```

8.3.3.3.3.2 The `connected=` parameter

This parameter expects a comparison filter, like the `sequence=` parameter. Its action is weaker, though. Using this parameter the data are sorted according to the criterion defined by the comparison filter, that is, according to the logic of a system of shelfmarks. The resulting index lines are, however, *not* combined to form a single line at the end.

8.3.3.3.3.3 The `continue=` parameter

This parameter condenses the resulting line even more strongly than the `sequence=` parameter. If two successive identifications contain at the beginning format elements which agree completely, only the divergent portions are displayed. With this parameter the lines

```
Huber, Julius:  123-12/4; farmer
Huber, Julius:  123-12/5; weaver
Huber, Julius:  242-3/1; innkeeper
Huber, Julius:  321-9/3; innkeeper
Huber, Julius:  321-9/4; weaver
Huber, Julius:  321-9/5; farmer
```

would therefore be reduced to the index line:

```
Huber, Julius:  123-12/4 f., 242-3/1, 321-9/3 - 9/5; farmer,
weaver, innkeeper
```

8.3.3.4 An index generated by several `index` commands

In most cases, a single `index` command is sufficient to create an index. However, there are certain situations where this is not possible. One example of such is provided by an index where for some groups a family name and for others a maiden name must appear in one and the same logical column. In such cases, you can formulate as many successive `index` commands as you like in one and the same $\kappa\lambda\epsilon\iota\omega$ task. Because of the way they work, this does not affect the meanings of the individual parameters, but you should note the two following points:

- You should only specify `part=` and `identification=` parameters for the second and subsequent `index` commands. $\kappa\lambda\epsilon\iota\omega$ does not actually prevent you from specifying other parameters for these commands, but in general you should avoid doing so: even at a later stage, only very experienced users should risk using others. There is no harm in simply copying the first command's parameters; in fact, this can be advantageous, since it facilitates the deletion of individual commands at a later date.
- The first `part=` / `identification=` parameter in *every* `index` command places an entry in the first logical column of the index; the second of these parameters in *every* `index` command places an entry in the second column etc. The user is responsible for ensuring that this results in columns containing terms which are easy to sort.

For the creation of such registers additional parameters exist. They have two purposes: (a) to clarify the sorting order between whole identical index lines, which shall be sorted differently, depending on which `index` command created them and (b) to differentiate between identical entries within a logical column, again depending on which `index` command created them.

8.3.3.4.1 Sort order of whole index lines

8.3.3.4.1.1 The `lines=` parameter

This parameter expects as a parameter value one of the two keywords `no` or `yes`. Both of these can be abbreviated to their respective first character.

It only makes sense if you use more than one `index` command to set up your indices; otherwise the system simply ignores it.

Where indices have been generated by more than one `index` command, a sort is not usually affected by which `index` command caused which term to be included in which logical column.

- `no` explicitly confirms this default setting.
- `yes` tells the system that two index lines in which the first logical column is regarded as identical should follow one another in the order corresponding to the ordinal number of the `index` command used to include the values in the logical columns. If, in a task in which you used two `index` commands, the first column on two lines matches and you set this parameter to `yes`, these two lines are printed in the same order as the commands which generated them, regardless of the contents of the following columns. So in this case, an index line generated by the first `index` command would appear before a line generated by the second, and so on.

In addition to `yes` or `no`, the `lines=` parameter accepts another value, consisting of any character which has not already been defined as a data signal character.

If you assign a character to the `lines=` parameter, the system uses it to create sort keys, on the basis of the logic described in the following section. You may only use this character as a parameter value if you define a `also=` parameter at the same time.

8.3.3.4.1.2 The `also=` parameter

The `also=` parameter expects a list of characters as a value.

You use it in conjunction with the `lines=` parameter described in the previous section, in order to perform complex sort operations based on terms. For this purpose, the system treats characters in the list of characters comprising this parameter's value as separators between further subdivisions contained in a logical column.

Its use can best be described with the help of an example.

Let us assume we have data in the form:

```
person$cleric=William (of York)
person$owner=William (of Folkstone)
```

We want to create a $\kappa\lambda\epsilon\iota\omega$ program capable of sorting clerics before other people with the same name, regardless of the contents of any designations of origin appearing in brackets.

The solution we adopt

```
index part=:cleric;lines=yes
index part=:owner
```

is not wholly successful, because the order of the two `index` commands only becomes relevant to the sort *after the end* of the logical column defined by the `part=` parameter.

By using the formulation

```
index part=:cleric;lines=+;also="("
index part=:owner
```

we achieve the required result, since the order of the `index` commands, which are now "in the middle of the logical column", becomes relevant to the sort.

This is achieved by writing an appropriate sort note at this point in the data, which the system limits by using the special character assigned to the `lines=` parameter. To enable the system to do this, you must make sure that it sorts the relevant logical column using a sorting order where you have assigned the same special character to the `conversion` directive's `form=` parameter as you assigned to the `lines=` parameter.

8.3.3.4.1.3 The `source=` parameter

This parameter expects a number as its parameter value. It provides for a further refinement of the model introduced by the `lines=` parameter.

In very complex applications it may become neccessary to specify explicitly in which order lines being produced by individual `index` commands shall be sorted, if they are otherwise identical. For this purpose an identification number can be assigned to the individual `index` command with this parameter. More than one `index` command can share one and the same number: lines produced by these `index` commands are than assumed to be of identical position in the sorting order.

If this parameter is used with some `index` commands within a $\kappa\lambda\epsilon\iota\omega$ task, it is highly recommended to specify it with each such command within this task.

8.3.3.4.2 Sort order of individual columns

The following mechanism provides for a very subtle difference in sorting individual columns, depending on the `index` command from which they are derived. We will first introduce the parameters which implements this mechanism and than explain it with an example.

The careful reader will notice that the problem solved here is (almost) identical to the one which was discussed with the `also=` parameter in section 8.3.3.4.1.2. The difference is, that with `also=` we have to rely on specific characters already present in the input data, while the present mechanism avoids this condition.

8.3.3.4.2.1 The `second=` parameter

This parameter expects a single character as parameter value.

To declare, that logical columns shall be sensitive for sorting according to the `index` command by which the index line containing it has been created, the `second=` parameter has to be specified excatly once. It should be assigned a character, which never occurs at the beginning of a line. This character will be inserted by $\kappa\lambda\epsilon\iota\omega$ for internal purposes at the beginning of some columns, but *not* occur in the output. (The character flags a logical column as "sensitive to explicit sorting".)

8.3.3.4.2.2 The `result=` parameter

This parameter expects a single character as parameter value.

The value of this parameter is inserted at the end of a column – *or replaces the first space character it contains* – before this column is compared for sorting purposes. The character does not appear in print.

8.3.3.4.2.3 Explanatory example

Let us assume we have data in the form:

```
person$cleric=William of York
person$owner=William of Folkstone
```

We want to create a $\kappa\lambda\epsilon\iota\omega$ program capable of sorting clerics before other people with the same name, regardless of the content of the field after the initial name.

The solution we adopt is:

```
index part=:cleric;secundum=~
index part=:owner;result=0
```

As a result, when the two names occur, the will be compared as

```
William of York
William0of Folkstone
```

so the cleric is correctly sorted before the secular person. (The zero does *not* occur in print.)

8.3.3.5 Examples

Below are some examples of correctly written `index` commands:

```
index part=:identifier;signs=10;
        part=:value;signs=5
```

```
index part=:identifier;type=count;
        part=:value;type=count
```

```
index part=:identifier;cumulate=yes;substitution='---"---';
        limit=" Value:  "
        part=:value;cumulate=yes;substitution='---"---';
        limit=" appears in:  "
```

8.3.4 The `create` command

The general form of the `create` command is:

NAME=<definition name>	**(MUST be specified!)**
PART=<list of paths>	**(MUST be specified!)**
OVERwrite=No | Yes	**(default: No)**
RESUlt=Yes | No	**(default: Yes)**
WRITe=No | Yes	**(default: No)**
IDENtification=<number>	**(default: 0)**
REPEat=No | Yes	**(default: No)**

It is used to generate codebooks which reproduce terms contained in a database. The system generally uses such codebooks to convert databases into files which can be subjected to statistical analysis; they can be coded for this purpose, i.e. provided with code numbers using the linguistic resources described in section 7.3.1.12.

When you are using this command to generate a codebook, you can include two variables for every identifier extracted from the database:

- the variable `kleio` numbers the identifiers included in the codebook in steps of ten, in the alphabetical order of the identifiers.
- the variable `system` is assigned the indicator for "missing values".

If you are supplementing an existing codebook by issuing another `create` command:

- no change is made to entries for identifiers which are already in the codebook.
- the following applies to the variable `kleio`:
  - no attempt is made to reintroduce it into the codebook if the user has removed it.
  - otherwise, a count in steps of ten is introduced for every newly introduced identifier. This count begins with the highest entry which was last made in this variable (multiplied by ten) and follows the alphabetical order of the *newly added* identifiers.
- all other variables for the newly added identifiers are set to the indicator for missing values.

8.3.4.1 The `name=` parameter

This parameter expects a user-defined name as a parameter value. It is used to set up a new codebook or to address one which you want to supplement.

8.3.4.2 The `part=` parameter

This parameter expects a list of paths as a value. The contents of all entries in this path which can be converted into the `text` data type are included in the codebook in the form of one identifier per entry.

8.3.4.3 The `overwrite=` parameter

This parameter expects one of the keywords `yes` or `no` as a parameter value.

- `no` means that when you set up a new codebook, there should be no codebook of the same name already associated with the database. If a codebook with the specified name does exist the system will refuse to execute the task. This is the system's default setting.
- `yes` instructs the system to destroy any existing codebook with the name you specified before setting up the new one.

8.3.4.4 The `result=` parameter

This parameter expects one of the keywords `yes` or `no` as a parameter value.

- `yes` instructs the system to try and set up a new codebook. This is the system's default setting.
- `no` instructs *κλειω* to add the new identifiers to an existing codebook. If no codebook with the name you specified exists the operation is aborted.

8.3.4.5 The `write=` parameter

This parameter expects one of the keywords `yes` or `no` as a parameter value.

- `no` instructs the system to include the extracted identifiers in the codebook with no further messages or prompts to the user. This is the default system setting.
- `yes` instructs *κλειω* to print out a list of the locations of all identifiers imported into the codebook. By default, the location of each identifier is designated by the identification of the document from which the identifier was taken.

8.3.4.6 The `identification=` parameter

This parameter expects a number as a parameter value. It tells the system how many levels away from document level it should search for a group identification with which to designate locations. In this case, the document is represented by zero, all groups contained in the document itself by one, and so on. If a suitable identifier is found in a group which is less than *number* levels away from the document, this group's identifier is used to identify the location instead.

Specifying this parameter implies you have also specified `write=yes`.

8.3.4.7 The `repeat=` parameter

This parameter expects one of the keywords `yes` or `no` as a parameter value.

- `no` instructs the system to terminate the task after entering the identifiers in the codebook. This is the system's default setting.
- `yes` instructs the system to output a description of the codebook contents in the form of a `describe` command once it has finished processing the command.

8.3.5 The `cumulate` command

The general form of the `cumulate` command is:

PART=<list of paths>	**(MUST be specified!)**
WRITe=<constant>	**(default: none)**
WITHout=Yes | No	**(default: Yes)**
RELAtion=No | Yes	**(default: No)**

It is used to create some simple statistics describing the parameters controlling the distribution of the contents of the entries specified by the `part=` parameter. These include the maximum value, minimum value, mean value, standard deviation and sum total of the distribution in question, together with the number of statistical cases included.

Hint: *this command is intended as a general aid to orientation. κλειω is not a statistics package. In almost every case involving more than the very simplest statistics, it is worth generating a file which can be subjected to statistical analysis by other means.*

8.3.5.1 The `part=` parameter

This parameter expects a list of paths as a parameter value. Every entry of the `number` data type in this list of paths is included in the specified calculations as a statistical case.

8.3.5.2 The `write=` parameter

This parameter expects an alphanumeric constant as a parameter value.

By default, κλειω prints out – on suitable output devices – the distribution requested in each case on a new page. If you specify this parameter, the character string you specify is printed out on the first line of each page.

8.3.5.3 The `without=` parameter

This parameter expects one of the keywords `yes` or `no` as a parameter value.

- `yes` means that in all cases where a `number` data item is expressed as an interval between a minimum and a maximum value, this entry is excluded from the calculation of statistical classification figures. (This is the system's default setting.)
- `no`, on the other hand, means that in such cases, the system calculates the mean value from the minimum and maximum values and weights its inclusion in the statistical classification figures using the following expression: $1 - \frac{(maximum - minimum)}{maximum}$.

8.3.5.4 The `relation=` parameter

This parameter expects one of the keywords `yes` or `no` as a parameter value.
It only has an effect if you are performing calculations over a network.

- `no` instructs the system to ignore an entry's position within a network. This is the system's default setting.
- `yes` evaluates the distance of each entry to be included in the calculation from the group in the database from which the first transition initiated by an element of the `relation` data type took place. Each entry is then divided by the square of its distance along the path. This means that an entry appearing in the same group as the original information item is divided by 1; an entry which was reached by making *one* transition via an element of the `relation` type is divided by 2; one which could only be reached by making *two* transitions is divided by 4, and so on. This option was included primarily for the purpose of weighting statistics in genealogical networks.

8.3.6 Creating statistical data records

$\kappa\lambda\epsilon\iota\omega$ makes two commands available so that the user can set up statistical data records.

The `translation` command defines which files the system should direct the results to; the `case` command defines how a subset of the information contained in a database should be transformed into a statistical case.

This enables the system to produce two kinds of result:

- a *data file* containing fixed-format cases, with (for the most part) numeric codes representing the variables.
- a *command file* containing the necessary commands for processing the data file in one of the two statistics software packages SPSS-X or SAS.

Since the way individual statistical cases are constructed is more important to an understanding of the way data is transferred to a statistics file than the way the surrounding conditions are defined, we shall start by explaining how they are constructed.

8.3.6.1 The `case` command

The general form of the `case` command is:

PART=<path> | <subpath>	**(MUST be specified!)**
MAXImum=<number>	**(default: one)**
REPEat=<number>	**(default: none)**
DATE=DAY | MONTh | YEAR | DATE | WEEKday | DIFFerence | NUMBer | AFTEr | BEFOre | DOUBle	**(default: DAY, MONTh, YEAR, DATE. Can be specified more than once.)**
NUMBer=Small | Large | Mean | Deviation | Number | Total	**(default: Number. Can be specified more than once.)**
SIGNs=<number-as-format>	**(default: numbers: 6.3; character strings: 20. Can be specified more than once.)**
NAME=<character string>	**(default: VARnnnn)**
WRITe=<character string>	**(default: path in part=)**
NULL=<number>	**(default: zero)**
WITHout=<number>	**(default: installation-dependent)**

A file for statistical analysis produced by $\kappa\lambda\epsilon\iota\omega$ consists of *cases* defined as one or more *case types*. Each case contains exactly the same number of variables in every case type: these variables can be analysed statistically.

Each `case` command causes exactly one of these case types to be generated. In many instances, one entry in a $\kappa\lambda\epsilon\iota\omega$ element will only generate a single statistical variable; frequently, however, it will also generate a whole group of such variables. Each of these (groups of) variables, which should all appear in the statistics file, is defined by a `part=` parameter. This group of variables can be described in more detail by subsequent parameters, which each relate to the preceding `part=` parameter.

8.3.6.1.1 Structuring the information

8.3.6.1.1.1 Simple cases

The `case` command causes one statistical case to be defined for every element encountered in the command's first `part=` parameter. If this element is missing from a group selected by the `query` command, the system does not generate a statistical case; if any elements specified in subsequent `part=` parameters are also missing, the system substitutes a *missing value* for the groups of variables to be derived from them.

Thus in the following example:

```
query name=source;part=person
case part=:age;
        part=:codebook[:occupation,status,occupations];
        part=:debts
```

a single case is generated for every person *whose* `age` *is known*, regardless of whether his/her `occupation` and/or `debts` are known.

If you want the system to generate a case for *every* person, you must use `form[]` – as for the `index` command – to ensure that the first element to appear is one which *always* appears, thus:

```
query name=source;part=person
case part=:form["0.0",number];
        part=:age;
        part=:codebook[:occupation,status,occupations];
        part=:debts
```

8.3.6.1.1.1.1 The `maximum=` parameter

If you do not add any further specifications to a `part=` parameter, cases are produced according to the following logic:

The first entry from every set of entries defined by the path mentioned in `part=` *is incorporated into the statistical case.*

The `maximum=` parameter means that the number of entries translated into independent variables or (e.g. in the case of `date`) groups of variables for each element selected in the preceding `part=` parameter, should be the same as the number assigned to `maximum=`.

```
case part=:codebook[:occupation,status,occupations];maximum=4;
        part=:date;date=month;date=year;maximum=2
```

consequently means that four variables are derived from `occupation`, each containing just one entry, and four variables are also derived from `date`, the first two containing the first entry's month and year, the second two containing the second entry's month and year, and so on.

8.3.6.1.1.1.2 The `repeat=` parameter

This parameter, which also expects a number as a parameter value, allows you to use another solution for the same problem. If you specify this parameter, the system only generates as many variables for each case as are necessary to process *one* entry; but it generates up to the <*number*> of cases you specified, each representing a single entry.

```
case part=:codebook[:occupation,status,occupations];repeat=4;
        part=:date;date=month;date=year;maximum=2
```

would consequently mean that the system generates one case for each of the first four entries in `occupation`, while deriving four variables for `date`, the first two containing the first entry's month and year, the second two containing the second entry's month and year.

The system repeats these last four variables for each of the cases derived from `occupation`.

If there are fewer entries than specified in `repeat=`, the system generates fewer statistical cases: i.e. if in our example, fewer than four occupational details appeared for a particular person, fewer cases would be generated.

8.3.6.1.1.1.3 Using `maximum=` to `repeat=` in comparisons

While you can combine these two parameters in any way you like, it is a good idea to keep to the following rules, for reasons related to content:

- `repeat=` only generates as many cases as are actually used; you can set as high a value as you like for this parameter.
- `maximum=` *always* generates the specified number of variables. For this reason, you should only request the number of variables you really need for your purposes: it will make statistical analysis very much more difficult if you create eight occupation variables simply because one person in twenty thousand happens to have eight occupations. In such cases, we recommend one of two solutions:
  - Either use the element function `cumulation[]` to create a variable which represents the number of existing occupational details but only includes the first two or three in the statistics file,
  - or combine (using a second codebook if necessary, regardless of the way you coded the individual occupations) the occupational details which appear into a single value using `collect[]`, which is then replaced by a code representing the particular configuration of the occupations in each case.
- If you use `repeat=` more than once in a `case` command, the individual cases are set up – as in the `index` command – so that the system uses every possible combination of the values of the groups of variables to which you assigned `repeat=` to create an case. Statistically speaking, files of this kind can be problematic, and should be avoided by beginners.

8.3.6.1.1.2 Subpaths in `part=`

As a rule, it is a good idea to use the `query` command to activate the group in `case` tasks which will act as the "case generator". If the case is supposed to contain a variable which is logically superior to the case-generating group, you should then activate this variable using `root[]` or `back[]`, as shown in the following example:

```
query name=source;part=person
case part=:codebook[:occupation,socialstatus,occupations];
        part=:age;
        part=root[0]:house_no
```

If this is not possible, so if a longer path is specified after `part=`, you may experience the following additional difficulty. According to $\kappa\lambda\epsilon\iota\omega$'s default definitions,

```
query name=source;part=house
case part=apartment/person:age
```

is defined as the set of all entries of all `age` specifications of all `persons` in all `apartments`. According to our definition of the `case` command above, only the first entry in this set is chosen. Thus the second and all subsequent apartments are ignored, together with the people living in them.

To avoid this problem, you can divide a path over several `part` parameters *in the* `case` *command only*. These parameters must appear in strict succession, with nothing separating them except for the `maximum=` or `repeat=` parameters used to specify the maximum number of groups in each case which should be brought in for processing. A `part=` parameter containing an element should appear at the end of this series of `part=` parameters with *subpaths*.

8.3.6.1.1.2.1 Using `maximum=` with subpaths

You use the `maximum=` parameter according to the logic described above.

```
case part=apartment;maximum=2;
        part=person;maximum=3;
        part=:codebook[:occupation,status,occupations];maximum=2
```

would accordingly instruct the system to generate the following statistical variables:

```
Variable 01:  Case number
Variable 02:  Apartment 1, Person 1, Occupation:  Entry 1
Variable 03:  Apartment 1, Person 1, Occupation:  Entry 2
Variable 04:  Apartment 1, Person 2, Occupation:  Entry 1
Variable 05:  Apartment 1, Person 2, Occupation:  Entry 2
Variable 06:  Apartment 1, Person 3, Occupation:  Entry 1
Variable 07:  Apartment 1, Person 3, Occupation:  Entry 2
Variable 08:  Apartment 2, Person 1, Occupation:  Entry 1
Variable 09:  Apartment 2, Person 1, Occupation:  Entry 2
Variable 10:  Apartment 2, Person 2, Occupation:  Entry 1
Variable 11:  Apartment 2, Person 2, Occupation:  Entry 2
Variable 12:  Apartment 2, Person 3, Occupation:  Entry 1
Variable 13:  Apartment 2, Person 3, Occupation:  Entry 2
```

8.3.6.1.1.2.2 Using `repeat=` with subpaths

The `repeat=` parameter offers a rather more dynamic solution to this problem. It instructs the system to begin a new statistical case every time it encounters the relevant group.

```
case part=apartment;maximum=2;
       part=person;maximum=3;repeat=yes;
       part=:codebook[:occupation,status,occupations];maximum=2
```

would accordingly result in the following data structure:

```
Case 1 Variable 01:  Case number
Case 1 Variable 02:  Apartment 1, Person 1, Occupation:  Entry 1
Case 1 Variable 03:  Apartment 1, Person 1, Occupation:  Entry 2
Case 2 Variable 01:  Case number
Case 2 Variable 02:  Apartment 1, Person 2, Occupation:  Entry 1
Case 2 Variable 03:  Apartment 1, Person 2, Occupation:  Entry 2
Case 3 Variable 01:  Case number
Case 3 Variable 02:  Apartment 1, Person 3, Occupation:  Entry 1
Case 3 Variable 03:  Apartment 1, Person 3, Occupation:  Entry 2
Case 4 Variable 01:  Case number
Case 4 Variable 02:  Apartment 2, Person 1, Occupation:  Entry 1
Case 4 Variable 03:  Apartment 2, Person 1, Occupation:  Entry 2
Case 5 Variable 01:  Case number
Case 5 Variable 02:  Apartment 2, Person 2, Occupation:  Entry 1
Case 5 Variable 03:  Apartment 2, Person 2, Occupation:  Entry 2
Case 6 Variable 01:  Case number
Case 6 Variable 02:  Apartment 2, Person 3, Occupation:  Entry 1
Case 6 Variable 03:  Apartment 2, Person 3, Occupation:  Entry 2
```

8.3.6.1.1.3 Using several `case` commands

During every task involving translation, the first variable in each case is reserved for a case number, each counted consecutively.

If you use several `case` commands in a task, the first `part=` parameters in all these commands always relate to the `part=` parameter in the preceding `query` command. Every command of this kind generates its own type of case. If $\kappa\lambda\epsilon\iota\omega$ encounters more than one `case` command, the second variable in all case types is reserved for the case type identifier. $\kappa\lambda\epsilon\iota\omega$ identifies the individual case types by the ordinal number of the relevant `case` command in the $\kappa\lambda\epsilon\iota\omega$ program. In the following example:

```
query name=source;part=family
case part=mother:age; ...
case part=father:age; ...
case part=child;repeat=yes;part=:age; ...
```

three case types are produced:

- Case type 1 for details about the father,
- Case type 2 for details about the mother and
- Case type 3 for details about the children.

If several types of case are generated, we would strongly advise you to include additional aids intended to help structure the resulting data and facilitate subsequent analysis, by instructing the system to output details of each document's ordinal number and/or details about the groups being processed.

8.3.6.1.2 Extracting information

The system derives one or more variables from each entry selected for processing by `part=`. The way this *group of variables* is constructed, and the format chosen for the output, depend on the data type.

8.3.6.1.2.1 The `number` data type

In the case of `number` data, the system only calls in entries in which a precise numerical value – i.e. not just a range between minimum and maximum values – is specified. This value is output to six places before and three places after the decimal point, and the latter is also included in the data. If there are not enough places for the number in front of the decimal point, the system strips off decimal places if possible. If the number is more than ten digits long, it is replaced by ten asterisks.

You can change the selection of information to be output using the `number=` parameter, and change the output format using the `signs=` parameter.

8.3.6.1.2.1.1 The `number=` parameter

This parameter expects as a parameter value one of the keywords `small`, `large`, `mean`, `deviation`, `number` or `total`, all of which can be abbreviated to their respective first character.

It specifies the variables which can be generated from elements of the `number` data type. You can specify this parameter as many times in succession as you need to in order to define the requisite group of variables.

By default the system generates one variable with a precisely known numerical value for every element of this data type. Again by default, the system treats elements for which only minimum and maximum values are known as missing.

More specifically, various keywords are used to instruct the system to include the following items of information in statistical variables:

- `small` instructs the system to output the minimum value if numerical data is inexact, and the exact number if it is exact.
- `large` instructs the system to output the maximum value if the numerical data is inexact, and the exact number if it is exact.
- `mean` instructs the system to output the mean value of minimum and maximum values if the numerical data is inexact and the exact number if it is exact.
- `deviation` instructs the system to output the difference between minimum and maximum values if the numerical data is inexact, or zero if the numerical data is exact.
- `number` instructs the system to output the number if the numerical data is exact, and to identify it as a missing value if it is inexact.
- `total` instructs the system to output the sum of minimum and maximum values if the numerical data is inexact, and the exact number if it is exact.

8.3.6.1.2.1.2 The `signs=` parameter

This parameter accepts any number as a parameter value. The system interprets the part of the number forming an integer as the number of digits appearing before the decimal point; the decimal fraction is interpreted as the number of decimal places required after the decimal point.

```
case part=:age;signs=2;
        part=:wealth
```

consequently instructs the system to output `age` as a two-digit number with no decimal places (and without a decimal point), `wealth` as a number with six places in front of and three places after the decimal point.

If the system encounters a number which cannot be displayed in this format, the format is modified using the procedure described above.

8.3.6.1.2.2 The `date` data type

By default, the system generates a group of three variables for every element of this data type, the first of which represents the day (two digits), the second the month (two digits) and the third the year (four digits) of a precisely known date. By default, the system regards elements for which only a time interval is known as missing.

You can modify the selection of data to be included using the `date=` parameter; you can change the output format using the `signs=` parameter.

8.3.6.1.2.2.1 The `date=` parameter

This parameter expects as a parameter value one of the keywords `day`, `month`, `year`, `date`, `weekday`, `difference`, `number`, `after`, `before` or `double`, all of which can be abbreviated to the *first four* letters of the keyword.

It defines which variables the system should generate from elements of the `date` data type. You can specify this parameter as many times in succession as necessary in order to define the required group of variables. If you specify this parameter once, the system starts by cancelling the default setting: the only data included in the statistical variables is what you request explicitly.

More specifically, various keywords are used to instruct the system to include the following items of information in statistical variables:

- Selecting separate items of information for the date:
  - `day` selects the day of the month (default output format: two-digit).
  - `month` selects the month (default output format: two-digit).
  - `year` selects the year (default output format: four-digit).
  - `weekday` selects the day of the week, in the following form: 1 = Monday, 2 = Tuesday, ..., 7 = Sunday (default output format: single digit).
  - `difference` selects the interval in days between the *terminus post* and *ante quem* (default output format: six-digit).
  - `number` selects the number of days which have passed since the first day of the first month of year one up to the day in question (default output format: six-digit).

- Choosing individual date formats:
  - `date` instructs the system only to use the selected information for dates which are known exactly.
  - `after` instructs the system to output the selected details for the *terminus post quem* where intervals of time are involved; or in the case of exact dates, to output details of the exact date.
  - `before` instructs the system to output the selected details for the *terminus ante quem* where intervals of time are involved; or in the case of exact dates, to output details of the exact date.
  - `double` represents a shortened form of `date=after;date=before`

  The following example:

```
case part=:date;date=year;date=double;
        part=:birthdate
```

hence generates an case which only contains the year, but does so for both date components for the element `:date`; for the element `:birthdate` on the other hand, it contains the default selection of day, month and year.

8.3.6.1.2.2.2 The `signs=` parameter

This parameter expects a number as a parameter value, which acts as a format specification.

The system interprets the part of this number which is an integer as the number of digits you wish to place in front of the decimal point; the part of the number following the (optional) decimal point as the number of decimal places required.

In the case of the `date` data type, *all* derived variables are always output in the given format if you specify this parameter. Because the default formats are generally speaking more memory-efficient, we advise you not to use this parameter for `date` data.

8.3.6.1.2.3 The `text` data type

If the system encounters `text` data in the element – which have not been transformed into `number` data by a codebook – the first twenty characters of the contents are incorporated into the statistics file as a character string by default, in the format of the platform in use.

You can use the `signs=` parameter to change this setting.

8.3.6.1.2.3.1 The `signs=` parameter

This parameter expects a number as a parameter value.

The part of the number which is an integer specifies the maximum number of characters in the preceding element which should be incorporated into the statistics file. Any data appearing after a decimal point is ignored:

```
case part=:name;signs=40;
        part=:firstname
```

hence means that forty characters from `:name` and twenty characters from `:firstname` should be included in the statistics file.

8.3.6.1.2.4 The `category` data type

The way this data is then processed for statistical purposes is regulated by the `category` definition you used to convert it. By appending a `number=` parameter to the `signs` directive, you can declare an explicit numerical value for each character, e.g. as follows:

```
item name=religion;usage=category
signs sign=c;write="Catholic";number=6
signs sign=p;write="Protestant";number=3
exit name=religion
```

If you do not use this parameter, the system generates the numerical values by counting up the individual `signs` directives beginning with one. If you had not specified the `number=` parameter in the example above, the values would have come out as `c = 1` and `p = 2`.

The system generates exactly the right number of variables for every `category` entry to ensure that

- every group of mutually exclusive characters defined by `part` is depicted in one variable and
- every character which can be combined with any other character, in any order, is depicted in its own variable.

We would like to point out that once you have established the linkage to a database, you can use a newly introduced `category` definition with the appropriate name to replace the temporary `category` definition held in the database. *But you when you do so, you must ensure that the individual characters are introduced in the same order as in the original definition.*

So in the following example, the system would ignore the `category` definition you used to set up the database:

```
query name=source;part=person
item name=religion;usage=category
part
signs sign=c;number=6
signs sign=p;number=3
exit name=religion
case part=:religion
```

The system uses a currently unmodifiable output format for `category` data: each derived variable is displayed with as many digits as necessary to represent the largest `number` value which appears in the codebook definition.

8.3.6.1.2.5 Other data types

At present, the system converts all other data types into the `text` data type before transforming them into statistical values.

8.3.6.1.3 The treatment of missing values

By default:

- the system replaces a variable representing a either a missing entry or element by zero.
- the "missing value" from a codebook is passed on directly to the statistics file.

8.3.6.1.3.1 The `null=` parameter

The `null=` parameter, which expects a number as its parameter value can be used to indicate either missing entries or elements with any other numerical value.

8.3.6.1.3.2 The `without=` parameter

The `without=` parameter, which expects a number as a parameter value, can be used to replace all missing values imported from a codebook with any other numerical value.

8.3.6.1.3.3 Example

```
case part=:codebook[:occupation,status,occupations];
            without=999;null=999;
        part=:codebook[:origin,region,placenames]
```

would consequently instruct the system to display any uncoded identifiers in the codebook as well as any missing entries as 999 in the case of `occupation`, but to restore the default settings in order to process `origin`.

8.3.6.1.4 Describing output in more detail

In the command file, the system identifies the different variables generated for the statistical software packages by assigning each variable a name formed from the prefix `VAR` and a four digit number in sequence.

The system also generates a command for every such variable, providing the variable with a label which explains its contents in more detail. By default, this label consists of the path which generated this variable, together with a suffix which specifies the kind of variable involved.

8.3.6.1.4.1 The `name=` parameter

This parameter expects as a parameter value a constant. When this parameter is specified, the constant is used to generate all variable names, which are derived from the element specified by `part`. If more than one such variable is created (as for example with `date` data, appropriate suffixes are added to the constant specified by the user.

So, the `case` command:

```
case part=marriage/child:dateofbirth;name="bdate"
```

will lead to the creation of the variable names `bdatepd`, `bdatepm` and `bdatepy`.

The suffixes are assigned as follows:

Data type `number`: A one character suffix, describing what kind of numerical information is put into the variable. The suffix is identical with the first character of the keyword with which the value is selected by the `number=` parameter.

`n` *number* (the default selection)
`s` *small* (or minimum)
`l` *large* (or maximum)
`m` *mean*
`d` *difference*
`t` *total*

Data type `date`: A two character suffix, describing which of the two calendar terms forming a $\kappa\lambda\epsilon\iota\omega$ date and which kind of date information is put into the variable. If no calendar term is selected, the suffix for the "earliest possible date" or *terminus post quem* is generated.

First part of suffix:

`p` terminus *post* quem or "earliest possible date".
`a` terminus *ante* quem or "last possible date".

Second part of suffix:

`d` *day* (part of the default selection)
`m` *month* (part of the default selection)
`y` *year* (part of the default selection)
`w` *weekday*
`i` *interval* (requested by *difference*)
`n` *number* of days

Data type `text`: No suffixes created.

Data type `category`: For each variable created from one element of this data type a one-character-suffix in alphabetical order, starting with the letter "A" is created.

8.3.6.1.4.2 The `write=` parameter

This parameter expects as a parameter value a constant. It replaces the path definition used to generate a label for the variable in question.

```
case part=:birthdate;name="dob";part=:dod
```

would thus mean that the system would assign the following tags to the variables derived from the element `birthdate` in the target statistical package (the example shows labels generated for SPSS-X):

```
VAR LABEL VAR0001 "case_no"
VAR LABEL VAR0002 "dob - day"
VAR LABEL VAR0003 "dob - month"
VAR LABEL VAR0004 "dob - year"
VAR LABEL VAR0005 ":dod - day"
VAR LABEL VAR0006 ":dod - month"
VAR LABEL VAR0007 ":dod - year"
```

8.3.6.2 The `translation` command

The general form of the `translation` command is:

FIRSt=<filename>	**(default: name of database + ”dat”)**
SECOnd=<filename>	**(default: name of database + ”dic”)**
OVERwrite=No | Yes	**(default: No)**
TARGet=SPSs | SAS | PCSpss | CENssys	**(default: SPSS)**
WRITe=Yes | No	**(default: Yes)**
CODEbook=<number>	**(default: 100)**
LIMIt=<number>	**(default: 100)**
DATE=DAY | MONTh | YEAR | DATE | WEEKday | DIFFerence | NUMBer | AFTEr | BEFOre | DOUBle	**(default: DAY, MONTh, YEAR, DATE. Can be specified more than once.)**
NUMBer=Small | Large | Mean | Deviation | Number | Total	**(default: Number. Can be specified more than once.)**
SIGNs=<number-as-format>	**(default: numbers: 6.3; character strings: 20. Can be specified more than once.)**
NULL=<number>	**(default: zero)**
WITHout=<number>	**(default: installation-dependent)**
MAXImum=<number>	**(default: one)**

This command describes the general framework of conditions in which a statistics file is generated. Each parameter has a default setting, so you can omit any or all of them if you wish.

A `translation` command controls subsequent `case` commands; if you need to, you can specify it as many times as you wish within a given task in order to impose different default settings on separate groups of `case` commands. In this case, every parameter value declared by this command remains in force until you explicitly cancel it. Hence both the following tasks would produce the same result:

```
query name=source; ...
translation first="statistics";second="commands";date=year
case ...
translation date=day;date=month;date=year;date=double
case ...
```

and

```
query name=source; ...
translation first="statistics";second="commands";date=year
case ...
translation first="statistics";second="commands";
        date=day;date=month;date=year;date=double
case ...
```

8.3.6.2.1 Defining the result files

By default, in the course of a task involving the creation of a file for statistical analysis, $\kappa\lambda\epsilon\iota\omega$ generates:

- A data file, composed of the database's name and the suffix ".dat". The name of the database is abbreviated as necessary so that it conforms to the maximum number of characters permitted on the platform in question; the suffix appears as a platform-specific filename extension: hence as a file extension under MS-DOS, as a file type under VM/CMS.
- A command file consisting of the name of the database and the suffix ".dic" (for data *dic*tionary). The database name is abbreviated so that it does not exceed the maximum number of characters permitted on the platform in question; the suffix appears as a platform-specific filename extension: so as a file extension under MS-DOS, as a file type under VM/CMS.
- If either the data file or command file already exist on the computer, the task is aborted.
- The command file contains commands for the statistical software package SPSS-X; the command file's suffix is `sps`.

You can change these settings using a preliminary group of parameters.

8.3.6.2.1.1 The `first=` parameter

This parameter expects a constant of any length as its definition. The system interprets this as the name of the data file you want to create.

The name may contain information about directories and suchlike in a form acceptable to your particular operating system.

Example for an MS-DOS system:

```
translation first="D:\stat\data.001"
```

8.3.6.2.1.2 The `second=` parameter

This parameter expects a constant of any length as its definition. The system interprets this as the name of the command file you want to create.

The name can contain information about directories and suchlike in a form acceptable to your particular operating system.

Example for an MS-DOS system:

```
translation second="D:\stat\format.001"
```

8.3.6.2.1.3 The `overwrite=` parameter

This parameter expects the keywords `no` or `yes` as a parameter value. Both of which can be abbreviated to their respective first character.

- `no` prevents the system from overwriting an existing file with the same name as the data or command file you specified, i.e. to behave as it would by default.
- `yes` allows the system to overwrite such files.

8.3.6.2.1.4 The `target=` parameter

This parameter expects as a parameter value one of the keywords `spss`, `pcspss`, `sas`, `censsys` or `null`. All of these can be shortened to the first *three* characters of each keyword.

- `spss` means that commands written to the command file are compatible with the SPSS-X statistical software package. A file handle statement for the input file is generated. The external filename produced for the input file may need editing on some mainframes, though. This is the system's default setting.
- `pcspss` means that commands written to the command file are compatible with the PC version of the SPSS statistical software package. A number of restrictions are taken care of: so only one missing value will be generated per variable; in codebook oriented processing this means that the separate missing value otherwise generated for terms in the database which are not in the codebook is *not* declared to be missing. The commands are separated by blank lines instead of being closed by full stops. Only one case type is allowed.
- `sas` means that commands written to the command file are compatible with the SAS statistical software package.
- `censsys` means that a subset of an SPSS data definition is produced, which is accepted by the CensSys tabulation program. Exactly two case types are required, which are assumed to be hierarchically related. Length restrictions on labels are relaxed.
- `null` suppresses the creation of the command file.

8.3.6.2.2 Defining other conditions

Regardless of the format you choose for the files to be generated, the following applies:

- $\kappa\lambda\epsilon\iota\omega$ represents the structure of the generated file in the form of a summary report.
- the system aborts the execution of a translation task if more than 100 entries of the `text` data type, which are supposed to be converted by a codebook, do not appear in the codebook in question and
- the system aborts the execution of a translation task if it finds more than 100 data items which it is unable to display in the specified format, hence must represent as asterisks in the data file.

These settings can be controlled by the following parameters

8.3.6.2.2.1 The `write=` parameter

This parameter expects one of the keywords `no` or `yes` as a parameter value. Both of these can be shortened to their respective first character.

- `yes` instructs the system to print out the file description generated by default.
- `no` instructs it not to do so.

8.3.6.2.2.2 The `codebook=` parameter

This parameter expects a number as a parameter value. It is interpreted as the maximum permitted number of codebook errors, i.e. of identifiers which cannot be found in the codebook in question.

8.3.6.2.2.3 The `limit=` parameter

This parameter expects a number as a parameter value. It is interpreted as the maximum permitted number of format errors, i.e. of values which cannot be displayed in the format in question.

8.3.6.2.3 Defining default settings for the `case` command

You can use the following parameters to specify a series of default settings for the `part=` parameter in all `case` commands which are *not* followed by this command's identically named parameters. If you used a `translation` command to specify one of these parameters earlier on, it remains as a default setting in the $\kappa\lambda\epsilon\iota\omega$ task in question until you explicitly cancel it by redefining this parameter in a new `translation` command.

8.3.6.2.3.1 The `number=` parameter

This parameter expects as a parameter value one of the keywords `small`, `large`, `mean`, `deviation`, `number` or `total`, all of which can be abbreviated to their initial character.

It tells the system which variables to generate in all the `part=` parameters of all `case` commands following this `translation` command from elements of the `number` data type, if the `case` command itself does not include a `number=` parameter. You can specify this parameter as many times in succession as you like in order to define the requisite group of variables. The meanings of the various keywords are explained in the detailed description in section 8.3.6.1.2.1.1.

8.3.6.2.3.2 The `date=` parameter

This parameter expects as a parameter value one of the keywords `day`, `month`, `year`, `date`, `weekday`, `difference`, `number`, `after`, `before` or `double`, all of which can be shortened to the *first four* letters of each keyword.

It instructs the system which variables to generate in all `case` commands' `part=` parameters, following this `translation` command, from elements of the `date` data type if the `case` command itself does not contain a `date=` parameter. You can specify this parameter as many times in succession as you like in order to define the requisite group of variables. The meanings of the various keywords are explained in the detailed description in section 8.3.6.1.2.2.1.

8.3.6.2.3.3 The `signs=` parameter

This parameter expects a number as a parameter value. The system interprets it as a format specification.

This parameter defines the standard output format. It may appear twice in a `translation` command.

If it appears *in front of* a `number=` parameter, it defines the default settings for the format of entries of the `text` data type. In this case, the system interprets the parameter value as described in section 8.3.6.1.2.3.1.

If it appears *after* a `number=` parameter, the system treats this format as the default setting for all variables derived from `number` elements. In this case, the parameter value is interpreted as described in section 8.3.6.1.2.1.2.

8.3.6.2.3.4 The `null=` parameter

This parameter expects a number as a parameter value. It specifies the numerical value which the system should use to display missing entries of the type described in section 8.3.6.1.4.1.

8.3.6.2.3.5 The `without=` parameter

This parameter expects a number as a parameter value. It specifies the numerical value which the system should use in a codebook to display uncoded identifiers of the type described in section 8.3.6.1.4.2.

8.3.6.2.3.6 The `maximum=` parameter

This parameter allows you to define a default value for complete paths and subpaths. The meaning of these definitions is discussed in sections 8.3.6.1.1.1.1 and 8.3.6.1.1.2.

Careful! If you use this parameter without some thought, the resultant files may become extremely large.

8.3.7 The `mapping` command

The general form of the `mapping` command is:

PART=<list of paths>	**(default: none)**
ALWAys=No | Yes	**(default: No)**
NAME=<name of group>	**(default: none)**
EACH=No | Yes	**(default: No)**
COLOur=Contrast | Red | Blue | Green	**(default: Contrast)**
LINEs=Simple | Double | Treble	**(default: Simple)**
USAGe=Solid | Halftone | Tenthtone | Empty	**(default: Solid)**
SYMBol=Circle | Square | Triangle	**(default: Circle)**
SIZE=<number>	**(default: device dependent)**
SIGN=Yes | No	**(default: Yes)**
NUMBer=No | Yes	**(default: No)**
WRITe=<character string>	**(default: none)**
FIRSt=<coordinate pair>	**(default: none)**
SECOnd=<coordinate pair>	**(default: none)**
TARGet=<file name>	**(default: see below)**
OVERwrite=No | Yes	**(default: No)**
WARNings=Yes | No	**(default: Yes)**

It is used to create maps.

A `mapping` command has two functions: First it selects a series of topographical objects to be displayed from a `location` definition, second, it defines the graphical attributes which these topographical objects should have on the map.

Topographical objects are selected for processing either by the fact that entries in their `location` format are included in the map from the database, or because a global instruction is issued to the effect that particular groups of objects contained in a `location` definition should be displayed.

All topographical objects selected for processing by a `mapping` command have the same graphical attributes: hence while it is sufficient to formulate a single `mapping` command in order to produce a map, it is normally necessary to issue several of them in order to display different classes of topographical object in various ways.

8.3.7.1 How several `mapping` commands interact

The way these different commands interact, best illustrated with the help of the following example, is determined by the following rules:

- You should only use `write=`, `first=` and `second=` parameters in the first `mapping` command.
- The `sign=`, `number=`, `colour=`, `usage=` and `lines=` parameters are all valid for each of the topographical objects selected by the `mapping` command in which they appear.
- If you use the `always=` and `each=` parameters in several `mapping` commands, only the parameter encountered in the last command is valid.

All of which brings us to the following example:

```
note All persons belonging to an abstract group are selected
note from the database -- e.g.  because they have a particular
note occupation.
query name=source;part=person:codebook[:occupation,status,occupations]=
        greater "299" and less "400"
note
note The map as a whole is labelled; all buildings which have not been
note selected by other commands are included on the map with their
note default attributes but without labels.
mapping write="Crafts and the Church";each=yes;sign=no
note
note Churches and monasteries are drawn on the map in green
mapping name=churches,monasteries;colour=green;colour=green;
        usage=solid;usage=tenthtone
note
note Craftsmen's homes are drawn in red
mapping part=root[0]:house_no;colour=red
```

8.3.7.2 Default settings for `mapping` commands

By default, every map you create using `mapping` contains

- all topographical objects from the `location` definition in which you specified a `always=yes` parameter and
- all topographical objects appearing in at least one of the entries addressed by `part=`.
- all topographical objects selected appear with the graphical characteristics defined for them in the `location` definition.
- the map expands to fill the drawing area as far as possible, and
- it is not labelled.
- if an output device has been defined by the `location` command, which leads to the creation of a file, that file is given a platform dependent name.
- if a file with that name exists already, the execution is aborted.
- for each topographical object in the entries addressed by `part=`, which cannot be found in the associated `location` definition, a warning message is generated.

8.3.7.2.1 Selecting topographical objects

8.3.7.2.1.1 The `part=` parameter

The `part=` parameter accepts a list of paths as a parameter value.

It selects all entries of the `location` data type addressed by that list for display on the map.

If the list of paths contains elements which relate to more than one `location` definition the system attempts to apply all `mapping` commands to every `location` definition addressed. This results in a series of maps, each one representing a `location` definition.

8.3.7.2.1.2 The `always=` parameter

This parameter expects one of the keywords `no` or `yes` as a value. Both of these can be shortened to their respective first character.

- `yes` instructing the system to include all topographical objects in the `location` definition with the parameter `always=yes` on the map, hence explicitly instructing the system to behave as it would by default.
- `no` prevents the system from including these objects on the map.

8.3.7.2.1.3 The `name=` parameter

This parameter expects a value consisting of a list of comma-delimited names, each of which appeared in the `name=` parameter of at least one `location` directive in the relevant `location` definition.

It instructs the system to include all topographical objects for which these names were specified on the map.

8.3.7.2.1.4 The `each=` parameter

This parameter expects one of the keywords `no` or `yes` as a value. Both of these can be abbreviated to their respective first character.

- `no` instructs the system not to display any topographical objects which are not explicitly included in the map by any of the other parameters listed in section 8.3.7.2.1. This is the system's default setting.
- `yes` instructs the system to include all topographical objects of the above type on the map.

8.3.7.2.2 Displaying topographical objects

8.3.7.2.2.1 The `colour=` parameter

This parameter accepts the keywords `contrast`, `red`, `blue` and `green` as values. All of these can be abbreviated to their respective first character.

It defines a colour for the topographical objects selected by the `mapping` command in which it appears. This colour may differ from the colour specified in the corresponding `location` definition. You can specify this parameter as often as you like.

The first `colour=` parameter you specify determines the colour of the topographical objects' contours. The second, which only applies to closed topographical objects, determines the colour of the fill. The third `colour=` parameter determines the colour of any secondary contours, i.e. of the first inner contour(s); the fourth parameter determines the colour of the area(s) enclosed by these contours, and so on.

The parameter values are interpreted as described in section 7.3.1.13.1.2.1.

8.3.7.2.2.2 The `lines=` parameter

This parameter accepts the keywords `simple`, `double` and `treble` as parameter values. All of these can all be shortened to their respective first character.

It determines the thickness of the contours of the topographical objects selected by the `mapping` command in question. This may differ from the value specified in the corresponding `location` definition.

The parameter values are interpreted in the manner described in section 7.3.1.13.1.2.2.

8.3.7.2.2.3 The `usage=` parameter

This parameter accepts the keywords `solid`, `halftone`, `tenthtone` and `empty` as parameter values. All of these can all be abbreviated to their respective first character.

It specifies a pattern for the topographical objects selected by the `mapping` command. This may differ from the pattern specified in the corresponding `location` definition. You can specify this parameter as often as you like.

The first `usage=` parameter you specify defines a pattern for the contours of the topographical objects. The second, which only applies to closed topographical objects, determines the area fill pattern. The third `usage=` parameter defines a pattern for secondary contours, i.e. the first inner contour(s); the fourth parameter defines a pattern for the area(s) enclosed by that contour, and so on.

The system interprets parameter values as described in section 7.3.1.13.1.2.3.

8.3.7.2.2.4 The `sign=` parameter

This parameter expects as a parameter value one of the keywords `no` or `yes`. Both of these can be abbreviated to their respective first character.

- `yes` instructs the system to label the selected topographical objects, regardless of the stipulations in the corresponding `location` definition.
- `no` suppresses these labels.

8.3.7.2.2.5 The `number=` parameter

This parameter expects as a parameter value one of the keywords `no` or `yes`. Both of these can be abbreviated to their respective initial character.

- `yes` instructs the system to specify how many times the topographical objects selected by `part=` appeared in the database.
- `no` prevents this, i.e. explicitly restores the system's default setting.

8.3.7.2.2.6 The `symbol=` parameter

This parameter accepts the keywords `circle`, `square` or `triangle` as parameter values. All of them can be shortened to their respective first character.

When this parameter is specified for a topographical object which consists of a single point, a symbol of the shape indicated by the keyword will be displayed centered around that point. If this parameter is specified, when more than one point is defined, the symbol will be displayed centered around the first point of the polygon describing the topographical object.

If a topographical object consists of only one point and this parameter is not specified, a circle will be displayed in its position.

8.3.7.2.2.7 The `size=` parameter

This parameter expects as value a number. It describes the default size of the symbol displayed for the topographical object. The number is interpreted according to the granularity of the individual graphical device on which the map is displayed. At the moment we recommend experimentation to find out about the most appropriate one.

8.3.7.2.2.8 The `cumulate=` parameter

This parameter expects as a parameter value one of the keywords `no` or `yes`. Both of these can be abbreviated to their respective initial character.

- `no` instructs the system to display a chosen value of the `symbol=` parameter always at the size specified by the `size=` parameter. This is the default behaviour of the system.
- `yes` displays the symbols depending on the frequency with which the topographical object occurs in the database. A topographical object occurring exactly once in the database will be displayed by the size defined by the `size=` parameter. Other topographical objects will be displayed in sizes where the displayed area is the logarithm to base ten of the observed frequency. That means: a frequency of ten is reflected by a symbol which fills twice the area of the symbol being displayed for an object occuring once, a frequency of hundred is reflected by a symbol which fills three times the basic area and so on.

8.3.7.2.3 The appearance of the completed map

8.3.7.2.3.1 The `write=` parameter

This parameter expects as a parameter value a constant which appears as a centred caption beneath the map generated, e.g. as follows:

```
write="Craftsmen's homes"
```

8.3.7.2.3.2 The `first=` parameter

This parameter expects as a parameter value two numbers separated by a comma.

The first number defines how far to the right of the left-hand edge of the drawing surface the map should start; the second specifies how far along the x-axis the drawing should extend.

8.3.7.2.3.3 The `second=` parameter

This parameter expects as a parameter value two numbers separated by a comma.

The first number defines how far below the top edge of the drawing surface the map should start; the second specifies how far along the y-axis the drawing should extend.

8.3.7.2.4 Files for preparing hardcopy

8.3.7.2.4.1 The `target=` parameter

If, with the help of the `location` command, a plotting device has been selected, which asks for the output of the `mapping` command to be stored in a file, the name of that file will depend on the system you are using. The most important types are:

- `kleio.plot` on all types of UNIX systems.
- `kleio.plt` on DOS systems.
- `kleio plot` on VM/CMS systems.

The `target=` parameter allows the user to override this filename. It expects as a parameter value a constant which is a file name according to the conventions of the specific operating system. Directory names, file types and the like can be specified up to any necessary length.

8.3.7.2.4.2 The `overwrite=` parameter

This parameter expects one of the keywords `yes` or `no` as a value. Both of them can be shortened to their respective first character.

- `no` instructs the system to protect the file which shall contain the output of the `mapping` commands against accidental overwriting. This applies to the default output file as well as to a file the name of which has been specified with the help of the `target=` parameter. This is the default behaviour of the system.
- `yes` allows the system to overwrite an existing file.

8.3.7.2.5 General options

8.3.7.2.5.1 The `warnings=` parameter

This parameter expects one of the keywords `yes` or `no` as a value. Both of them can be shortened to their respective first character.

- `yes` instructs the system to issue a warning message for each topographical object which becomes eligible for display, because it is included in the `part=` parameter of the `mapping` command, cannot be found in the associated `location` definition, however. This is the default behaviour of the system.
- `no` suppresses these messages.