6. Converting raw data into a database

Data are always read into a $\kappa\lambda\epsilon\iota\omega$ data base from an ASCII file. Such ASCII files can either be prepared by the menu driven input facilities described in volume B10 of the "Halbgraue Reihe" or with any text editor capable of storing pure ASCII files.

6.1 General

You use the `read` command to convert input data into a database. As soon as the system identifies this command, it automatically switches into another input mode. For this reason the system does not interpret the first line with an empty first column as the end of the command, but the first line containing data signal character 1.

If you have issued the `read` command, and it has been diagnosed as error-free, the system expects you to enter data immediately afterwards. Consequently the system will misinterpret any commands issued or definitions made after a `read` command.

`read` expects input data to start with a new document. If this is not the case, the system skips over a given number of other groups (the exact number depends on your particular installation) and then aborts the $\kappa\lambda\epsilon\iota\omega$ program if it is still unable to find a document.

A `read` task is terminated by one of three possible situations:

- If the system comes to the end of the data after the `read` command, it exits from the database and returns control to the operating system.
- If the system encounters a

  ```
  stop$
  ```

  group in the input data after the `read` command, it exits from the database at this point, ignores any other data in the currently active input file and returns control to the operating system.
- If the system encounters a

  ```
  continue$
  ```

  group in the input data after the `read` command, it exits from the database at this point and awaits any further instructions from the currently active input device. The user may, for example, wish to redefine a logical object in order to use a different currency conversion rate for a different part of the source, and then instruct the system to continue to create the database by entering another `read` command. Or the user may decide to move straight on to an analysis routine for the database which has been created.

Instead of using the `stop` or `continue` symbols you may also use the various resources described in section 5.5.

6.2 The `read` command

The general form of the `read` command is:

NAME= <filename> **(MUST be specified!)**
REPEat= No | Yes **(default: No)**
LINEs= <number> **(default: value of the same parameter in the currently active Options)**
WRITe= Names | Parts | Structure | Generic **(default: no report; can be specified several times over)**
CONTinuation= Clio | Null | <special character> **(default: Clio)**
CUMUlate=<special character> **(default: none)**
SUBStitution=<special character> **(default: none)**
MORE=<special character> **(default: none)**
NULL=<special character> **(default: none)**
FORM=No | Yes **(default: No)**

6.2.1 Referring to a database

Every `read` task must refer explicitly to an existing database. This database may be the result of a preceding structure definition (hence empty) or an earlier `read` task.

6.2.1.1 The `name=` parameter

The `name=` parameter expects the name of an existing database as a value.

6.2.2 Controlling the scope of the printed output

By default, while $\kappa\lambda\epsilon\iota\omega$ is converting the data into its own format, it only prints out the processed data if this appears to be necessary in order to illustrate an error. It also prints out a brief set of statistics on the total number of information units contained in the database, once the conversion procedure is complete.

6.2.2.1 The `repeat=` parameter

The `repeat=` parameter accepts the keywords `yes` or `no` as values. Both keywords can be abbreviated to their respective first character.

- `no` instructs the system not to print out the processed data, i.e. to behave as it would by default.
- `yes` instructs the system to print out the processed data, with no space lines between each line of data.

6.2.2.2 The `lines=` parameter

The `lines=` parameter expects a number between zero and four as a value. This parameter tells the system how many space lines to insert between every two lines of the imported and printed input data. If you specify this parameter, the currently active value of the `options` command's `lines=` parameter is restored once this particular `read` task has been completed.

6.2.2.3 The `write=` parameter

The `write=` parameter accepts the keywords `names`, `parts`, `structure` or `generic` as values, all of which can be abbreviated to their respective first character. You can specify this parameter as often as you like, using a different keyword every time. (This should be in the format `write=names;write=parts` *not* `write=names,parts`). This allows you to define any extra printed output with considerable precision.

- `names` instructs the system to print out a list of descriptions of all names in the database which would be valid element names,
- `parts` instructs the system to print out a list of descriptions of all valid names of groups in the database,
- `structure` instructs the system to print out a description of the structural relationships between all non-generic identifiers within the database, and at the same time to specify any structural restrictions (minimum / maximum permitted frequencies) and
- `generic` instructs the system to print out a list of descriptions of all names in the database which would be valid generic names (for elements (*and* groups).

6.2.3 Defining input control characters

$\kappa\lambda\epsilon\iota\omega$ processes the imported data as a set of information fields structured by data signal characters. Within these fields individual characters are assigned a meaning according to the data type involved.

You can modify this behaviour so that individual characters are assigned a meaning which does not depend on data type and which controls the way the imported data is converted before it is broken down into separate information units and analysed in greater depth.

6.2.3.1 The `continue=` parameter

The `continue=` parameter accepts the keywords `clio`, `null` or any other individual character which is not a letter or a number.

- `clio` instructs the system to ignore blank spaces at the end of every line of input and to join the next line to the previous one automatically in such a way that the last character on the first line (which is not an empty space) appears immediately in front of the first character on the second line. This is how the system behaves by default.
- `null` instructs the system to ignore blank spaces at the end of every line of input and to join the first character on the second line (which is not an empty space) to the last character on the first line after inserting a single blank space between the two characters.
- If you define a special character (a *line continuation character*), you can instruct $\kappa\lambda\epsilon\iota\omega$ to check whether the last character at the end of every line is the same as this special character. If it is, the system removes this special character and any preceding spaces, and the character appearing immediately in front of it is linked directly to the first character on the next line. If it is not, the system behaves as it would if you had specified the keyword `null`.

6.2.3.2 One way to simplify input

In order to process a list-based source laid out as follows:
Hubert Müller, 35 years old, a carpenter from Neuberg
Crescentia, his wife, 33 years old
Their son Hubert, a 14-year-old carpenter's apprentice
Their daughter Anna, 11 years old
Their daughter Walpurga, 8 years old
Friedrich Hübner, Crescentia's uncle, in retirement in his 64th year

any details transferred from one person (= group) to the next must be explicitly repeated, in accordance with what we have said above. As follows, for example:

```
head-of-household$Hubert/Müller/35/carpenter/Neuberg
wife$Crescentia/Müller/33
son$Hubert/Müller/14/carpenter's apprentice
daughter$Anna/Müller/11
daughter$Walpurga/Müller/8
relative$Friedrich/Hübner/64/retired/relationship=uncle/
    centre-of-family=wife
```

The following format would have corresponded more closely to normal orthographical conventions:

```
head-of-household$Hubert/Müller/35/carpenter/Neuberg
wife$Crescentia/"/33
son$Hubert/"/14/carpenter's apprentice
daughter$Anna/"/11
daughter$Walpurga/"/8
relative$Friedrich/Hübner/64/retired/relationship=uncle/
    centre-of-family=wife
```

You can declare this abbreviated form of input using the `cumulate=` and `substitution=` parameters in the `read` command.

There are no default settings for either of these parameters. They are interpreted as follows:

6.2.3.2.1 The `cumulate=` parameter

If, at the input stage, an element contains the special character which you have assigned to the `cumulate=` parameter, and only this character, it is replaced by the contents of the *element* with the same name which was last encountered in the database. Thus all entries of all aspects are repeated. If an element starts with this character but it is then followed by any other character the system suspects an error and issues a message. If an element contains this particular character in any other position than the initial one, it is treated and processed like any other character.

Thus this parameter allows you to input data just as illustrated above.

However, it always relates to *entire elements*. Hence the following attempt to repeat an item of conceptual information while also appending an additional commentary, would *not* be possible using the `read` command

```
read name=example;cumulate="

head-of-household$Hubert/Müller/35/carpenter/Neuberg
wife$Crescentia/"/33
son$Hubert/"#son by first marriage with ?/14/carpenter's apprentice
daughter$Anna/"/11
daughter$Walpurga/"/8
relative$Friedrich/Hübner/64/retired/relationship=uncle/
    centre-of-family=wife
```

6.2.3.2.2 The `substitution=` parameter

The `substitution=` parameter is used for this purpose.

This behaviour of this parameter depends on whether the entries in the element concerned are subordinated to the aspects or vice versa.

6.2.3.2.2.1 `substitution=` in "simple" entries

If no option has been specified, or if the user used a `element` directive with the parameter `order=simple` in the element concerned, $\kappa\lambda\epsilon\iota\omega$ regards the entries as a further subdivision of the aspects, as we have already seen. Each element then has exactly three aspects, each one of which can be subdivided into as many entries as you like. The following definition applies to elements of this type:

If one aspect consists of a special character which was previously assigned to the `read` command's `substitution=` parameter, this character is replaced by all entries in the aspect which has just been processed in the element of the same name which was last encountered in the input data. If another character follows it within the same aspect, the system suspects an error and issues an error message. If an element contains the particular character in a position other than the first position of an entry it is treated and processed just like any other character.

If the last specified element of the same name does *not* contain the aspect which is to be repeated, the repetition character is ignored and an error message displayed. So if we specify the special character assigned to `substitution=` after the comment character "`#`" in an element, but there was no comment in the last-encountered element of the same name, no attempt is made to find a comment in an element even further back; instead no repetition is made.

The following solution is appropriate for the example just quoted:

```
read name=example;cumulate=";substitution=&
head-of-household$Hubert/Müller/35/carpenter/Neuberg
wife$Crescentia/"/33
son$Hubert/&#son by first marriage with ?/14/carpenter's apprentice
daughter$Anna/&/11
daughter$Walpurga/"/8
relative$Friedrich/Hübner/64/retired/relationship=uncle/
    centre-of-family=wife
```

If from the preceding definitions, it is *not* clear to you why the following solution would have an undesirable side effect (pay particular attention to `Anna Müller`), we would suggest that you enter both examples into your computer and compare the results of a `write` printout *before* attempting to work with the feature.

```
read name=example;cumulate=";substitution=&
head-of-household$Hubert/Müller/35/carpenter/Neuberg
wife$Crescentia/"/33
son$Hubert/&#son by first marriage with ?/14/carpenter's apprentice
daughter$Anna/"/11
daughter$Walpurga/"/8
relative$Friedrich/Hübner/64/retired/relationship=uncle/
    centre-of-family=wife
```

6.2.3.2.2.2 `substitution=` **in "multiple" elements**

If you have used an element containing a `element` directive with an `order=multiple` parameter, the entries will divide up the aspects, i.e. each one of these elements consists of any number of entries, each one of which includes up to three aspects. The following definition will then apply to `substitution=`:

If an entry consists of a special character which was assigned to the `read` command's `substitution=` parameter, it is replaced by all aspects of the entry with the same ordinal number in the last element of the same name encountered in the input data. If another character follows it within the same aspect, the system suspects an error and displays an error message. If an element contains this particular character in any other than the first position of an entry it is treated and processed just like any other character. If the last-specified element of the same name does *not* contain an entry with the corresponding ordinal number the repetition character is ignored and an error message displayed. Thus if an element containing two entries is followed by an element containing three entries followed by a repetition character this character is ignored. No attempt is made to find an earlier element containing a fourth entry.

6.2.3.3 Support for fixed-format fields

In order to support the option for defining “fixed-format” input fields described in more detail in section 4.5, you can use the `read` command to define two characters which allow you to input and manage such data more efficiently.

The following parameters have no effect on the processing of elements which have not been assigned a fixed length: if the characters defined in this way are encountered in another element they are also processed just like any other character.

6.2.3.3.1 The `null=` parameter

The `null=` parameter expects any character which has not already been reserved as a data signal character as a parameter value.

If an element which has been assigned a fixed length consists exclusively of this character it is treated as if it did not exist.

6.2.3.3.2 The `more=` parameter

The `more=` parameter expects a parameter value consisting of any character which has not already been reserved as a data signal character.

If this character is encountered in an element for which a fixed length has been declared, as the first character in the first entry, it is regarded as introducing a repetition instruction as described in section 4.5.3.1.

6.2.4 Complex redefinitions of input and output

As a rule, it makes sense to preserve input data in separate files. The data can then be read from these files by means of a redirecting `options` command without further ado. However, the disadvantage of this approach is that the `read` command must be written at the beginning of each file containing input data. In order better to resolve this and a range of other problems, you can arrange things so that once the `read` command's vitally necessary parameters have been analysed – for the time being these include the `name=` parameter – and any other parameters required by the user have been processed, the `read` command's remaining specification is processed as if it was an `options` command. You do this by specifying a `form=` parameter.

The `form=` parameter accepts either of the keywords `yes` or `no` as a specification. They can be abbreviated to their respective first character.

- `no` causes the system to treat the remainder of the specification as if it was part of the `read` command; that is, prompts the system to behave as it would by default.
- `yes` causes all parameters following this particular parameter to be interpreted as if they were part of an `options` command (cf. chapter 7 in this manual). Please note that parameters with the same name in a `read` command are in certain circumstances interpreted differently from the way they are interpreted in an `options` command. Thus the value of the `lines=` parameter in a `read` command is cancelled out once this command has been processed. In contrast the value of this parameter is preserved if it is encountered after `form=yes`, hence interpreted as part of an `options` command.

6.3 Examples

The following are examples of correctly written `read` commands:

```
read name=source
```

```
read name=source;repeat=yes;form=yes;
      source="data";first=1;limit=17;part="dat"
```