Data with daTeX

Lenstra, A.; Kliffen, S.; Koning, Ruud

Published in:
EPRINTS-BOOK-TITLE

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
1995

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):
Lenstra, A., Kliffen, S., & Koning, R. H. (1995). Data with daTeX. In EPRINTS-BOOK-TITLE s.n..

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Download date: 25-12-2018
Data with $\delta a_{\TeX}$

Andries Lenstra, Steven Kliffen and Ruud Koning
lenstr@sci.kun.nl

Abstract
The authors explain how to handle data in TeX documents, in particular, how to avoid ever having to type in – and check! – the same data or text twice. These data may be stored in ordinary (non-TeX) databases, in ASCII files arranged according to the easy $\delta a_{\TeX}$ format, or in the TeX document itself. $\delta a_{\TeX}$ works in plain TeX and is supposed to work in \LaTeX.

1 Introduction
As soon as one uses the same data more than once, or the same text with different data, a classical problem arises: how far should one go in leaving the repetitions to the machine? $\delta a_{\TeX}$, a collection of TeX macros for storing and retrieving data, is an engine that enables one to go all the way, so that document source texts are as short as possible and integrity of the data is guaranteed, as well as uniformity in typesetting.

That TeX is suitable for repetitive tasks such as mail merges, was already shown e.g. in [2]; that a more general approach is also feasible, and that, in fact, TeX is an ideal word processor for data handling, is what we hope to show here.

The key feature of TeX, of course, is its programmability in plain, readable text, so that in the first place the manipulating c.q. processing of data and the perfect fine-tuning of text to data are easy and rewarding tasks – certainly also for non-Texicilians, as practice has shown. One could say that the personalizing of printing, or data handling in general, opens up a whole new realm of applications for the power of TeX as a programming language. Now this beautiful machinery not only generates the typography, but prior to this the very text. Examples can be found in [2] and [1].

In the second place the existence of expandable, plain text macros has great benefits in the set-up and maintenance of databases. Such macros can be a substitute for real, explicit data, and their expansion can be changed according to different circumstances. Imagine, for instance, someone who uses TeX for the typesetting of concert programmes in several languages. In his database of music pieces, the contents of the field ‘key’ belonging to some piece in A major (some music pieces have a coordinate called key) ‘A
major' is a key) will not be A major, but something like Amaj, which auxiliary files will translate into 'A major', 'A Dur', 'La majeur', or 'A grote terts', etc., dependent on the desired language. Or think of the printing of a price list: the contents of the field 'price' need not be the explicit digits of the price, but may expand into a formula from which the explicit digits will follow after a calculation with adjustable parameters.

This possibility of filling the contents of a data base with 'meta'data, \TeX macros that expand to the actual, printed data, can even be exploited further. Consider in the first example the field 'composer' (most of the music pieces have a composer). Instead of putting there a specific name, e.g. Mozart, or W.A. Mozart, or W.A. Mozart (1756--1791), why not put the command

\Some Later To Be Specified Set Of Fields
Belonging To The Entry 'Mozart2'
(in \TeX abbreviated as \*Mozart2* *). If there is a second database, of composers, with an entry that is identified by the string Mozart2, and \TeX is told where to find this entry? - Then all questions as to the inclusion of initials and dates can safely be left unanswered until the time of typesetting. Besides, apart from being more versatile and less error-prone, in the presence of this second database such an approach is mandatory because of the problem of data integrity: as soon as the same explicit data are entered more than once, how can one be sure that a change or correction is carried through in all occurrences in all documents?

One will realize that with this method of letting fields of one entry refer to other entries by means of commands built around an entry-identifying string, one can give shape to many relationships between data, in a readable and easily memorized way, so that with minimal effort a coherent, structured set of data is obtained. The workings of \TeX, which uses the idea of the identifying string, are now readily sketched.

2 Sketch of \TeX

With \TeX, the referencing facility also exists in ordinary document source text. Suppose \TeX has learned \TeX and that one writes

\Of \*Mozart2* * {\bf Name, \Initials} (\TownOfBirth)

in the document. Suppose also that earlier in the document, or in a separate data file, the following lines occur:

**Mozart2* \Name Mozart; \Initials W.A.; \TownOfBirth Salzburg;

\The string \*Mozart2* * is an abbreviation for
\Of \*Mozart2* * \Type

\Mozart, W.A. (Salzburg)
The string \*Mozart2* * is an abbreviation for
\The string \*Mozart2* * is an abbreviation for
\Of \*Mozart2* * \Type
The Type is a field that every entry should have; for Mozart one could choose Type to be Composer:

\texttt{***Mozart* \ldots Type \textbackslash Composer; \textbackslash Year\textbackslash Of\textbackslash Birth 1756; \ldots}

Then Type will expand to Composer and Composer acts as a template, say

\texttt{\textbackslash def\textbackslash Composer\{%}
\texttt{\{bf \Name, \Initials\}
\texttt{\{\textbackslash Year\textbackslash Of\textbackslash Birth, \textbackslash Year\textbackslash Of\textbackslash Birth\}\%
\texttt{\}}%

With this definition of Composer the typeset result of writing *Mozart* will be

\textbf{Mozart, W.A. (Salzburg, 1756)}

Instead of writing

\texttt{Of *Mozart* \textbackslash Year\textbackslash Of\textbackslash Birth}

(with a space after the final *) in order to get 'Salzburg', one can also write

\texttt{\textbackslash def\textbackslash Composer\{\textbackslash Town\textbackslash Of\textbackslash Birth\} *Mozart* *}

The latter may seem too much if it is used only once, but imagine the possibilities if the referencing is repeated. For instance, with these three templates

\texttt{\textbackslash def\textbackslash M\textbackslash piece\{\textbackslash Comp\}\%
\texttt{\textbackslash def\textbackslash Composer\{\textbackslash Town\textbackslash Of\textbackslash Birth\}\%
\texttt{\textbackslash def\textbackslash Town\{\textbackslash Country\}\%

the result of *(some music piece)* will be the country where one can find the town where the composer was born, assuming that one filled the proper fields in the proper databases with the proper three-starred identifying strings c.q. data. In the next section we will do this explicitly as an illustration of the \LaTeX{} format for storing data.

For repetitive tasks, however, it is immediately clear that the template is the perfect tool. A template is put implicitly in the document by *(an identifying string)* as above; writing three Composer

\texttt{*Bach* *}
\texttt{*Mozart* *}
\texttt{*Beethoven* *}

(or as one line \texttt{*Bach* **Mozart* **Beethoven* *}) gives three times the expansion of Composer, in the first case separated by spaces. This repetition is done automatically by \texttt{\textbackslash Filter}. In \LaTeX{} the \texttt{\textbackslash Filter} command filters a data file according to a certain Type, i.e. it considers every entry in consecutive order and, if the Type corresponds to the given Type, e.g. Composer, it expands the template, in this case Composer. So this is the way to perform mail merges, in which a form letter, the template, is merged with a data file that contains the data of the persons that are supposed to receive the letters. The exact description and an example can be found in Section 7.

Explicitly asking for one or more fields of a certain entry, as in

\texttt{The composer \texttt{Of *Mozart* * \textbackslash Initials\{} \texttt{Name\{} ...}

At that time, \texttt{\textbackslash Year\textbackslash Of\textbackslash Birth}, the town \texttt{\textbackslash Town\textbackslash Of\textbackslash Birth\{} was ...
follows the rule that should be obvious from the use of \$\Def{}$: all field names give the contents belonging to the last identified entry, which means that as soon as \$\Def{} \* \text{(a different identifying string)} \* \$ appears, \$\Def{} \* \text{Beethoven} \* \$ say, all field contents are of the new entry. Beethoven; those of Mozart2 are forgotten. If Mozart2 has a field \$\YearOfBirth{}\$, but Beethoven has not, then after the appearance of \$\Def{} \* \text{Mozart} \* \$ the text \$\Def{} \* \text{Beethoven} \* \text{\YearOfBirth}{} \$ will cause the error message

\text{Use of} \$\YearOfBirth{}\$ \text{doesn't match its definition.}

and if the same text is not preceded by any \$\Def{} \* \text{(string identifying an entry which has a field \$\YearOfBirth{}\$)} \* \$, then it will cause the error message

\text{! Undefined control sequence.}

With the help of the $\LaTeX$ command \$\IfField{}\text{\YearOfBirth}{} \text{Exists}\$ such embarrassing moments can be avoided. Tools like this one can be found in Section 5.

Under circumstances to be explained later, it will save time to tell \$\LaTeX{}$ in which database it should look for the desired identifying string. Such a specification may take the place of the space between the second and the third *, as in

\begin{verbatim}
*\text{Mozart2}c:/music/data/comp.dat*
\end{verbatim}

or in

\begin{verbatim}
\def\Dpath\text{c:/music/data/}\
\Def{} \* \text{Mozart2} \* \Dpath \text{comp}\* \YearOfBirth
\end{verbatim}

the extension .dat being supplied by $\LaTeX$ if no extension has been specified.

### 3 Storing data

$\LaTeX$ has its own way of storing data, the $\LaTeX$ format. In order for data to be accessible to $\LaTeX$, they should be stored according to this format, for instance in a $\LaTeX$ file. The user of $\LaTeX$ may use his own data base programs as long as they are capable of producing intermediate ASCII files; these are the subject of Section 6. As soon as the composition of the identifying string has been specified, the conversion to the $\LaTeX$ format can be taken care of by $\LaTeX$.

Data base programs other than $\LaTeX$ often offer facilities, such as sorting, that until now for $\LaTeX$ files only exist in cooperation with such a data base program via an intermediate file, or with the operating system. (Sorting inside $\LaTeX$ is possible; see [3].) The $\LaTeX$ format, on the other hand, is very simple, versatile, and easy to learn, while $\LaTeX$ has a very powerful sorting-out mechanism. Files according to the $\LaTeX$ format are as portable as ordinary $\TeX$ document source text files; in fact, they can be integrated, wholly or partially, into the document, as we shall see.

In the $\LaTeX$ format data are stored in data blocks. A $\LaTeX$ file has a name with a one, two, or three character extension, but not necessarily with the extension .dat, and consists of a number of data blocks. Before and between these data blocks one should put only \text{\NoDefaults} and \text{\Default} commands (to be described shortly) or comments.
Data with \TeX

and after the last data block a \TeX file should be empty – at least until one exactly knows what is going on.

A data block consists of a number of entries. An entry starts with \texttt{**} (the identifying string)* followed by any number of fields and occurrences of this three-starred identifying string. A field consists of optional spaces followed by a control word denoting the field name (a control word is of the form \textbackslash (a string of letters without spaces)) followed by the field contents followed by a semicolon.

\begin{verbatim}
(field name) (field contents);
\end{verbatim}

and an entry ends with the occurrence of \texttt{**} (a different identifying string)*, with which a new entry starts, except when this last string is \texttt{zzzz}. The string \texttt{zzzz} is supposed to be different from all strings used for entries; after \texttt{**zzzz} the data block ends. The fields may be broken by the identifying strings, the entry minus all occurrences of \texttt{**} (the identifying string)* should be a sequence of fields. So

\begin{verbatim}
**Mozart2*\Name Mozart;\Type\Composer; \ChristianNames Wolfgang\%  
**Mozart2* Amadeus ;**Beethoven* \Name Van Beethoven;\Type \Composer
; **Beethoven***Salzburg*\Type\Town;\Country Austria;
**zzzz
\end{verbatim}

is a data block, albeit a somewhat untidy one. The rationale for allowing the identifying strings breaking the fields is that \texttt{**} (the identifying string)* should occur at the beginning of every line of the entry and nowhere else, so that no two entries share the same line. For such data blocks some manipulations with the data are possible with the help of common outside tools. Before we discuss these possibilities, let us clean up the above data block, adding some data and \texttt{\Defaults}.

\begin{verbatim}
\NoDefaults
\Default\Type\Composer;
\Default\Name\Ident;
**Mozart2*\Name Mozart;\TownOfBirth *Salzburg* *;
 \ChristianNames Wolfgang\%
**Mozart2* Amadeus ;  
**Mozart2* \YearOfBirth1756; \YearOfDeath 1791;
**Beethoven* \Name Van \Ident;
**Bach*
**Salzburg*\Type\Town;\Country Austria;
**KV 488*\Type\Mpiece;\Key \Amaj;\Comp *Mozart2* *;
**zzzz
\end{verbatim}

Now with the definitions from the previous section \texttt{KV 488*} will indeed give ‘Austria’. The control word \texttt{\Ident} always expands to the identifying string of the entry. For the entry \texttt{Bach two fields exist} (by \texttt{\Default}), the field with field name \texttt{Name} and contents \texttt{\Ident}, and the field with field name \texttt{Type} and contents \texttt{\Composer}. In the entry \texttt{Mozart2} the contents of the field with field name \texttt{ChristianNames} are
Wolfgang Amadeus, with a space at the end. Without the %-sign there would have been two spaces between these names.

In practice, one probably would not mix up so many different \Types in one data block, and one would add a list of field names for every \Type, just to be sure that no such things happen as using together \Forenames and \ChristianNames. But one has the complete freedom to invent new fields on the spot and to list them in any order, independent from other entries, as long as all fields are closed by the delimiter ; (semicolon). This should be carefully checked, apart from the explicit data themselves.

Vice versa, as soon as a semicolon appears, chances are high that it terminates the field contents. If one needs the semicolon in the contents of a field, one should use \Semicolon. The typeset asterisk, '*', is available as \Star.

The choice of the identifying string is free, as long as it identifies the entry (i.e. all entries have different strings), is not empty, and only contains letters, digits, spaces, and no two spaces in a row. \TeX will only check this to a limited extent.

Whole, closed data blocks can be put anywhere in the document. A \Filtered file can be absorbed, i.e. memorized, searched for an entry, or \Filtered; a data block in the document is always absorbed.

### 3.1 Manipulations with data blocks

If all entries of a data block have their **(identifying string)** at the beginning of every line and nowhere else, then sorting the block on the first column means sorting the data on the identifying string. This is useful in its own right and also a means of bringing together the different parts of an entry that is scattered around the block. A way of taking care of broken fields would be by inserting sorting dummies like \zzz1, \zzz2, \defined as {}.

Moreover, with the help of the grep command, well known to Unix users, one can sort on field contents by having \Filter make a list of these contents paired with the identifying strings.

\[(\text{possibly processed) field contents} \; \text{(identifying string)}\]

sorting the list on field contents, and asking grep to rearrange the data block according to the new order of the identifying strings.

Finally, grep allows preprocessing of \TeX documents in which \TeX is invoked. The searching of files for strings by grep will be faster than by \TeX, so that having grep search the document for three-starred identifying strings, search the \TeX files for the lines on which these occur, and putting these lines in the document (taking care of the proper \Defaults and the closing of the data block by **zzzz**) will sometimes save time.
4 System set-up

For simple \TeX tasks there is nothing left to learn except the use of a few tools, and the fact that the size of data blocks in documents is limited because they are absorbed (how much \TeX can absorb depends on the local implementation). If \TeX has to search unspecified \TeX files, however, \TeX has to know these files and may need some guidance in their treatment. The

```
\TheDataFiles\firstfile\cdots\InSearchOrder
```

command tells \TeX which \TeX files should be absorbed (\((x)=(a)\)), which \TeX files should be searched if no search-file has been specified after the second star (\((x)=(a)\) or \((x)()\)) and in what order they should be searched, and which of these default search-files get a 'search-only' treatment (\((x)=(a)\)) under \DefaultMemoryProtection.

Whenever

```
*(an identifying string)\cdots\textup{optional \TeX file specification}*
```

or

```
\Or\ *(an identifying string)\cdots\textup{optional \TeX file specification} \cdots\textup{field name}
```

invoke \TeX to retrieve data, \TeX will normally try to remember these. If the data block was absorbed in which the identifying string occurs, or if \TeX has looked up the data already once before, it will succeed in searching its memory and find the data. Otherwise the data are looked up in the search-file specified after the second star or, if there is no such file, in the default search-files, i.e. all non-absorbed files of the list of \TheDataFiles.

So the normal way is that the more strings \TeX looks up, the more data it will remember. This means that when \TeX is invoked for many different strings, \TeX may run out of memory. Therefore \DefaultMemoryProtection allows for something special if a \TeX file has been specified in the list as (a), 'search-only', and has been specified between the second and third star, as in *(an identifying string)\cdots\textup{name of a search-only \TeX file}*, then \TeX will not try to remember the data but will look them up immediately in the specified file, use them, and forget them.

In the \DefaultMemoryProtection mode the protection of the memory has to be activated by specifying a search-only file after the second star. However, in the \StrongMemoryProtection mode the memory is always protected: the only way to have \TeX search its memory immediately is by specifying an absorbed, (a), file after the second star. In this mode the specification of any non-absorbed file makes \TeX act as after the specification of a search-only file in the default mode. If there is no specification at all, \TeX will not try to remember the data (this would involve the forming of a control sequence, and the number of control sequences that \TeX can see in a single run is limited), but search the default search-files for them. If it finds the data, it uses them and forgets them; in the absence of success it will conclude that the data must have been absorbed and only then search its memory.

\TeX always starts in \DefaultMemoryProtection but the user can alternate between this mode and \StrongMemoryProtection.
4.1 File specification

When between the second and third star an absorbed file or a search-only file is specified, taking the place of the space, then in order for \alpha\LaTeX\ to be able to recognize this file as absorbed or search-only, it is not only necessary that the \texttt{TheDataFiles} command has been executed already, but also that the ‘canonical’ file name of this file on this place in the document is the same as when it was listed in the list of \texttt{TheDataFiles}.

The canonical file name is the result of the following reduction: \alpha\LaTeX\ expands all control sequences in the typed-in file name, e.g. \texttt{Dpath in Dpath comp} in Section 2, then it tries to remove possible spaces at the beginning and at the end, and checks if the result has an extension, i.e. ends on .x or .xy or .xyz, with x, y, and z letters. If there is an extension then this is preserved, otherwise the extension .dat is attached. So always writing the same name is by no means necessary, but for the same \alpha\LaTeX\ file one should not switch between file name with full path included, and file name.

If the expansions of the control sequences in the typed-in file names do not begin or end with a space and there are no + signs in file names, then there will most probably be no problems with spaces around the file names or around the delimiters (a), (s), and (.). For instance, one can write *(identifying string)\texttt{Dpath comp} * as well as *(identifying string)\texttt{Dpath comp.dat} and

\begin{verbatim}
\texttt{TheDataFiles}
\texttt{Dpath comp.dat (a)}
\texttt{Dpath mpiece(s)town ()}
\texttt{InSearchOrder}
\end{verbatim}
as well as

\begin{verbatim}
\texttt{TheDataFiles}
\texttt{Dpath comp.dat (a) Dpath mpiece (s)}
\texttt{town () InSearchOrder}
\end{verbatim}

for that matter, provides a check-list.

5 Tools

5.1 Default fields

The \texttt{NoDefaults} and \texttt{Default} commands, introduced in Section 3, may only be given outside a data block, so when one wants to change the default contents of a field, or wants to add a default field, then one should first put **zzzzz**. The syntax of \texttt{Default} is

\begin{verbatim}
\texttt{Default(field name) (default field contents); (space)}
\end{verbatim}

which is the same as \texttt{Default (default field) (space)}. The necessary space after the semicolon could be provided by giving every \texttt{Default} a line of its own. \alpha\LaTeX\ puts the default fields immediately behind the first **(identifying string)** of an entry in the given
order (new default fields behind the old ones), before the fields that are explicitly typed in. The contents of a field are overridden by those of a subsequent field with the same field name. This holds for all fields, default or explicitly typed in, so that by
\default{same field name} (new default field contents);(space)
one can change the default contents of a field. \tex starts with an empty list and will not change this list unless it is told to do so by a \default or \nodef command.
In order to know the exact contents of this list at every moment and to avoid surprise results when \tex absorbs or searches a sequence of \dat files, one should have it emptied often by the \nodef command – for instance at the beginning of every \dat file.

5.2 Conditionals
We introduce three \if commands. Like \tex's ordinary \if..., they have an optional \else part, are closed with \fi, and may be nested. \tex allows \fi instead of \fi. All examples refer to the second data block of Section 3.

The definition of a control word can be inspected by the command
\ifCs{control word}\isDefinedAs{(a string)}
After \<* (an identifying string)* the control word \ident is defined as (this identifying string), so that the typeset result of
\\{bf\}Name\}
\fi
is 'Mozart'. With \<* (any other identifying string)* the result will be nothing.

The possibility of checking on the existence of fields was announced already in Section 2. The condition
\ifField(field name)\exists
is true for all entries for the field names \Type and \Name, e.g.
\{bf\}Name\}
\fi\else\* \ifField Name\exists is true;
\fi\else\* \ifField TownOfBirth\exists is false.

Finally, here is a facility for testing if an existing field looks like a given field:
\ifExistingField\looksLike(field name) (given field contents);
In this comparison the field contents are left untouched by \tex, i.e. they are not expanded or processed otherwise. So the typeset result of
\{bf\}Name\}
\fi\fi
\fi
is 'Mozart'. With \texttt{Def \*(any other identifying string from our data block)}\$* the result will be nothing. Furthermore:

\begin{verbatim}
\texttt{Def \*Mozart2* \texttt{IfExistingField\ LooksLike \texttt{Name Mozart}; is true.}
\texttt{Def \*Beethoven* \texttt{IfExistingField\ LooksLike\ Name Van \texttt{Ident}; is true.}
\texttt{Def \*Bach* \texttt{IfExistingField\ LooksLike \texttt{Type Composer}; is true.}
\end{verbatim}

The comparison of a given string with processed field contents is a different matter. Consider, for instance, the string Van Beethoven. This string is equal to the result of processing the field contents \texttt{Name Van \texttt{Ident}} in the following way: expand all that is expandable until there is nothing expandable left. After \texttt{Def \*Beethoven* \* the process of producing out of \texttt{Name} the field contents \texttt{Van \texttt{Ident}} themselves, also only involves expansion. Therefore, for the control word \texttt{Name} the effect of

\begin{verbatim}
\texttt{Def \*Beethoven* \* \texttt{edef Name\{Name\}}%}
\end{verbatim}

is the same as

\begin{verbatim}
\texttt{edef Name\{Van Beethoven\}%}
\end{verbatim}

All other fields \texttt{Name} in our data block also have contents that can be expanded completely, i.e. until there is nothing expandable left. This means that in this case we are back to the \texttt{IfCst} technique learnt above. The typeset result of

\begin{verbatim}
\texttt{Def \*Beethoven* \* \texttt{edef Name\{Name\}}%}
\texttt{IfCst Name\texttt{IsDefinedAs\{Van Beethoven\}}%}
\{bf Name\}%
\end{verbatim}

is 'Van Beethoven', and with \texttt{Def \*(any other identifying string)}\$* the test will work but the result will be nothing. (We could have used \texttt{Name} itself instead of \texttt{Name}, but then the protection against misuse of the field name \texttt{Name}, as exposed in Section 2, would have disappeared.)

Field contents of the form \*(an identifying string)\$* cannot be expanded completely; they cannot be put in an \texttt{edef} without \TeX{} having to stop and complain, or behaving in some other undesirable manner. If in our data block there had been an entry of which the \texttt{Name} had contents of this form, the above test would not work properly and the \texttt{IfExistingField\ LooksLike} test should be preferred.

Now for processed field contents where the processing involves more than expansion. The comparison of

1. field contents that refer to other entries, but of which the stage that is to be compared is completely expandable, with
2. strings in which nothing expandable is left,

is possible with \texttt{edef (=/global/edef)}. Consider, for instance, the string Austria, the result of \texttt{KV 488*} with the templates of Section 2. After \texttt{Def \*Salzburg* \*}, the process of producing Austria out of \texttt{Country} only involves expansion, but after \texttt{Def \*KV 488* \* the process of producing Austria out of \texttt{Comp} involves more than sheer expansion, and
Data with \TeX

\begin{verbatim}
\def \KV 488* * \edef \cDef{\Comp}{\Comp}%
\end{verbatim}
is not recommended. The \Comp field contents \emph{Mozart2*} refer to other entries, but the stage \Country, that is to be compared, is completely expandable. The solution is to have \TeX do its work, but put the result aside for later testing, wrapped in a control word: when

\begin{verbatim}
\def \Town{\xDef{\Country}{\Country}} \KV 488* %
\end{verbatim}
hence

\begin{verbatim}
\IfCs \Country \IsDefinedAs {\Austria} %
\end{verbatim}
is true. Of course, more would be needed to make the test work for arbitrary \Mpieces. Now the \Comp and all other fields have to exist down to \Country.

One can also test immediately and export the result. This will be shown for strings and field contents that should not be touched. If the \Country field contents are \emph{\Austria*}, then after

\begin{verbatim}
\newif{\IfRight}
\def \Town{\global \IfRightfalse}
\ifExistingField \LooksLike \Country \Austria* * ; %
\global \IfRight \True
\fi
\end{verbatim}

it will be seen that \IfRight is true.

\subsection*{5.3 Wrapping}

In the last test only the result of the comparison was available, not a control word like \Country for further testing or processing. As an analogue to \xDef, the \WrapIn command provides this facility whenever \TeX should not touch the field contents to be compared. After \Def{\pseudo}{\string} \WrapIn\{ield\}

\begin{verbatim}
\WrapIn \(\pseudo\field\) \field
\end{verbatim}
is equivalent to

\begin{verbatim}
\def \(\pseudo\field\) {\{\field\content\}}
\end{verbatim}

Here is the penultimate example again; the \Country field contents are assumed to be \emph{\Austria*}. The condition in

\begin{verbatim}
\def \Town{\WrapIn \Country \country} \KV 488* %
\IfCs \Country \IsDefinedAs {\Austria* * %
\end{verbatim}
is true, and \Country is available for other purposes.

\section*{6 Conversion}

Most data base programs are capable of reading and producing ASCII data files. \TeX offers two utilities that facilitate the cooperation with such programs. The first reads almost all ASCII data files that the latter may produce, and converts these data files
into δαΤ files. The second converts, for almost every data base program, a δαΤ file into the particular ASCII format that is readable for the program. These utilities are currently under construction.

7 Filtering

The \Filter command, announced in Section 2, filters a δαΤ file according to a certain \Type:
\Filter (δαΤ file specification)\Type (the filter type)
for instance.
\Filter \Dpath comp \Type\Composer
considers every entry of \Dpath comp in consecutive order and, if the \Type is equal to \Composer, it expands \Composer. δαΤX reduces the δαΤ file specification to the canonical file name, as explained in Section 4. A space before '\Type' should not cause any problems.

The restriction to one \Type is not essential. Suppose one has a big δαΤ file mail.dat with many entries of different \Types sorted on some postal code. If all entries should receive a letter, then \Filtering the δαΤ file once for every \Type would destroy the ordering. The solution is to replace, in mail.dat, all occurrences of '\Type' by '\ProType' (i.e. to change the field name \Type into \ProType), to add, at the top of mail.dat, the line
\Default\Type\FormLetter;
if there is one template, \FormLetter, or the line
\Default\Type\ProType;
if every old \Type has its own template, and to
\Filter mail.dat \Type\FormLetter
or to
\Filter mail.dat \Type\ProType
respectively. For detailed examples of form letters we refer to [1].

We conclude with an example that filters a δαΤ file of \Composers. If there is a field \TownOfBirth, its contents refer to a δαΤ file of towns, i.e. are of the form *(a town)*. Every \Town has \Default \Name \Ident, and a \Country with contents that are macros to be translated by a translation file:
\def\Town{\xdef\Name{\Name}\WrapIn\cDntry\Country}\% 
\def\Composer{\% 
  \IfField \TownOfBirth \Exists 
  \TownOfBirth 
  \IfCa\cDntry\IsDefinedAs\{Austria\}\% 
  \{\bf\Name}, from \Name, \cDntry\par
  \Fi 
  \Fi 

\% Filter c:/dat/composer
The result is a list of all composers in c:/dat/composer.dat that are born in Austria, with their towns of birth.

References
[2] M. Piff. Text merges in \TeX and \LaTeX. Taken from the file textmerge.dtx provided with the program source code, April 21, 1995.