Data with $\delta\alpha\text{T\kern-.1667em E\kern-.125em X}$

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 $\delta\alpha\text{T}$ format, or in the \TeX\ document itself. $\delta\alpha\text{T}$ 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? $\delta\alpha\text{T}$, 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-\TeX\icians, 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 A maj, which auxiliary files will translate into 'A major', 'A Dur', 'La majeur', or 'A grote tert', 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

```latex
\texttt{\textbackslash \texttt{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

```latex
\texttt{\textbackslash \texttt{\textbar Name, \textbar Initials}} (\texttt{\textbar TownOfBirth})
```

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

```latex
\*\texttt{Mozart2* \textbar Name Mozart; \textbar Initials W.A.;\textbar TownOfBirth Salzburg;}
```

Then the typeset result will be

```latex
\texttt{Mozart, W.A. (Salzburg)}
```

The string *Mozart2* * is an abbreviation for

```latex
\texttt{\textbar Of *Mozart2* * \textbar Type}
```
The \Type is a field that every entry should have; for Mozart one could choose \Type to be \Composer:
\Type \Composer; \YearOfBirth 1756; 
Then \Type will expand to \Composer and \Composer acts as a template, say
\Composer %
{\bf \Name, \Initials}
(\TownOfBirth, \YearOfBirth) %
With this definition of \Composer the typeset result of writing \Composer * will be
\Composer
\Composer (Salzburg, 1756)
Instead of writing
\Composer * \TownOfBirth
(with a space after the final *) in order to get 'Salzburg', one can also write
\Composer \TownOfBirth \Composer *
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
\Mpiece{\Composer %}
\Composer {\TownOfBirth} %
\Town{\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 \TeX 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 \Composers
*\Composer*
*\Composer*
*\Composer*
(or as one line \Composer **\Composer **\Composer ) gives three times the expansion of \Composer, in the first case separated by spaces. This repetition is done automatically by \Filter. In \TeX the \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
*\Composer\Of \Composer \Composer \Composer\Of \Composer\Of \Composer\Of \Composer\Of \Composer\Of \Composer
At that time, \YearOfBirth, the town \TownOfBirth was ...
follows the rule that should be obvious from the use of \texttt{\textbackslash Or}: all field names give the contents belonging to the last identified entry, which means that as soon as \texttt{\textbackslash Or \textbackslash* (a different identifying string) \* \* appears, \texttt{\textbackslash Or \*Beethoven \* \* say, all field contents are of the new entry, Beethoven; those of Mozart2 are forgotten. If Mozart2 has a field \texttt{\textbackslash YearOfBirth}, but Beethoven has not, then after the appearance of \texttt{\textbackslash Or \*Mozart2 \* \* the text \texttt{\textbackslash Or \*Beethoven \* \* \textbackslash YearOfBirth} will cause the error message

\texttt{Use of \textbackslash YearOfBirth\textbackslash Or doesn't match its definition.}

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

\texttt{! Undefined control sequence.}

With the help of the \texttt{\textbackslash TeX} command \texttt{\textbackslash IfField\textbackslash YearOfBirth\textbackslash 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 \TeX{} 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

\texttt{\*Mozart2+c:/music/data/comp.dat*}

or in

\begin{verbatim}
\def\Path{c:/music/data/}
\texttt{\textbackslash Or \*Mozart2+\Path comp* \textbackslash YearOfBirth}
\end{verbatim}

the extension .dat being supplied by \texttt{\textbackslash TeX} if no extension has been specified.

\section{Storing data}
\texttt{\textbackslash TeX} has its own way of storing data, the \texttt{\textbackslash TeX} format. In order for data to be accessible to \texttt{\textbackslash TeX}, they should be stored according to this format, for instance in a \texttt{\textbackslash TeX} file. The user of \texttt{\textbackslash TeX} 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 \texttt{\textbackslash TeX} format can be taken care of by \texttt{\textbackslash TeX}.

Data base programs other than \texttt{\textbackslash TeX} often offer facilities, such as sorting, that until now for \texttt{\textbackslash TeX} files only exist in cooperation with such a data base program via an intermediate file, or with the operating system. (Sorting inside \TeX{} is possible; see [3].) The \texttt{\textbackslash TeX} format, on the other hand, is very simple, versatile, and easy to learn, while \texttt{\textbackslash TeX} has a very powerful sorting-out mechanism. Files according to the \texttt{\textbackslash TeX} 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 \texttt{\textbackslash TeX} format data are stored in data blocks. A \texttt{\textbackslash TeX} 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 \texttt{\textbackslash NoDefaults} and \texttt{\Default} commands (to be described shortly) or comments,
and after the last data block a \texttt{Tag} 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.

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}
**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{Mozart}2 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 \Type s 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. \LaTeX will only check this to a limited extent.

Whole, closed data blocks can be put anywhere in the document. A \LaTeX 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 `zz1. zz2. \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.

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

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 \LaTeX is invoked. The searching of files for strings by \grep will be faster than by \LaTeX, so that having \grep search the document for three-starred identifying strings, search the \LaTeX 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
\begin{verbatim}
\DataFiles (first file name)(x) ... \InSearchOrder
\end{verbatim}
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)=(0)) and in what order they should be searched, and which of these default search-
files get a 'search-only' treatment ((x)=(0)) under DefaultMemoryProtection.

Whenever
\begin{verbatim}
*(an identifying string)*(optional \TeX{} file specification)*
\end{verbatim}
or
\begin{verbatim}
\OOf *(an identifying string)*(optional \TeX{} file specification)* \{field name
\end{verbatim}
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
\DataFiles.

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)*\{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 \TeX 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: \TeX 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 \TeX 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 \texttt{*[identifying string]* \Dpath comp *} as well as \texttt{*[identifying string]*\Dpath comp *\Dpath comp dat (a) \Dpath mpiece(s)town ()}

\texttt{InSearchOrder}
as well as

\texttt{TheDataFiles}

\texttt{\Dpath comp.dat (a) \Dpath mpiece (s) town () \InSearchOrder}

as \texttt{TheDataFiles}, 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 \texttt{*zzzzz*.} The syntax of \texttt{Default} is

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

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. \TeX puts the default fields immediately behind the first \texttt{**(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{\text{same field name}} \langle \text{new default field contents} \rangle ; \langle \text{space} \rangle
one can change the default contents of a field. \LaTeX{} starts with an empty list and will not change this list unless it is told to do so by a \default{} or \nodefult{} command. In order to know the exact contents of this list at every moment and to avoid surprise results when \LaTeX{} absorbs or searches a sequence of \LaTeX{} files, one should have it emptied often by the \nodefult{} command—for instance at the beginning of every \LaTeX{} file.

5.2 Conditionals

We introduce three \texttt{\texttt{If}}... commands. Like \TeX{}'s ordinary \texttt{\texttt{if}}...\texttt{s}, they have an optional \texttt{\texttt{else}} part, are closed with \texttt{\texttt{fi}}, and may be nested. \LaTeX{} allows \texttt{\texttt{Fi}} instead of \texttt{\texttt{fi}}. All examples refer to the second data block of Section 3.

The definition of a control word can be inspected by the command
\texttt{\texttt{IfCs}}\texttt{\texttt{control word}}\texttt{\texttt{IsDefinedAs}}\langle \text{a string} \rangle
After \texttt{\texttt{Of}} *(an identifying string)* * the control word \texttt{Ident} is defined as \langle this identifying string \rangle, so that the typeset result of
\begin{verbatim}
\texttt{\texttt{IfCs\{\texttt{control word}\}\texttt{IsDefinedAs}\langle \text{a string} \rangle}}
\end{verbatim}
\begin{verbatim}
\texttt{\texttt{Of \texttt{*\{an identifying string\} \texttt{\{control word\}\texttt{IsDefinedAs}\langle \text{a string} \rangle}}}  \texttt{\texttt{is \texttt{\langle this identifying string \rangle}, so that the typeset result of}}}
\end{verbatim}
\begin{verbatim}
\texttt{\texttt{Of \texttt{*\{Mozart\} \texttt{IsDefinedAs}\langle \text{Mozart} \rangle}}}  \texttt{\texttt{is \texttt{Mozart}. With \texttt{Of \texttt{*\{any other identifying string\} \texttt{\texttt{\{control word\}\texttt{IsDefinedAs}\langle \text{a string} \rangle}}} \texttt{the result will be nothing.}}}
\end{verbatim}

The possibility of checking on the existence of fields was announced already in Section 2. The condition
\begin{verbatim}
\texttt{\texttt{IfField\{field name\}\texttt{Exists}}}
\end{verbatim}
is true for all entries for the field names \texttt{\texttt{Type}} and \texttt{\texttt{Name}}, e.g.
\begin{verbatim}
\texttt{\texttt{Of \texttt{*\{Bach\} \texttt{IfField\{Name\} Exists}} \texttt{is true;}}}
\end{verbatim}
\begin{verbatim}
\texttt{\texttt{Of \texttt{*\{Bach\} \texttt{IfField\{TownOfBirth\} Exists}} \texttt{is false.}}}
\end{verbatim}

Finally, here is a facility for testing if an existing field looks like a given field:
\begin{verbatim}
\texttt{\texttt{IfExistingField\{field name\} \texttt{\texttt{\{given field contents\};}}}
\end{verbatim}
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
\begin{verbatim}
\texttt{\texttt{Of \texttt{*\{Mozart\} \texttt{IfField\{TownOfBirth\} Exists}}}
\end{verbatim}
\begin{verbatim}
\texttt{\texttt{IfExistingField\{\texttt{TownOfBirth\} LooksLike \{\texttt{Salzburg\};}}}
\end{verbatim}
\begin{verbatim}
\texttt{\texttt{\{\texttt{bf\{Name\}}\};}}
\end{verbatim}
\begin{verbatim}
\texttt{\texttt{Fi}}
\end{verbatim}
\begin{verbatim}
\texttt{\texttt{Fi}}
\end{verbatim}
is ‘Mozart’. With \IfDef*{any other identifying string from our data block} * * the result will be nothing. Furthermore:
\IfDef*{Mozart} * \IfExistingField\LooksLike\Name Mozart; \IfDef* true.
\IfDef*{Beethoven} * \IfExistingField\LooksLike\Name Van \Ident; \IfDef* true.
\IfDef*{Beethoven} * \IfExistingField\LooksLike\Name Van Beethoven; \IfDef* false.
\IfDef*{Bach} * \IfExistingField\LooksLike\Type Composer; \IfDef* true.

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 \Name \Ident in the following way: expand all that is
expandable until there is nothing expandable left. After \IfDef*{Beethoven} * the process
of producing out of \Name the field contents \Ident themselves, also only involves
expansion. Therefore, for the control word \Name the effect of
\IfDef*{Beethoven} * \edef\Name{\Name} is the same as
\edef\Name{Van Beethoven}.

All other fields \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 \IfCs technique learnt above. The typeset result of
\IfDef*{Beethoven} * \edef\Name{\Name}
\IfCs\Name isDefinedAs{Van Beethoven}
{\bf\Name}
\If
is ‘Van Beethoven’, and with \IfDef*{any other identifying string} * * the test will work
but the result will be nothing. (We could have used \Name itself instead of \Name, but
then the protection against misuse of the field name \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 \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 \Name had contents of this form, the above test would not work properly
and the \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 \edef (=\global\edef). Consider, for instance, the string Austria,
the result of *KV 488* * with the templates of Section 2. After \IfDef*{Salzburg} * * ,
the process of producing Austria out of \Country only involves expansion, but after
\IfDef*{KV 488} * * the process of producing Austria out of \Comp involves more than
sheer expansion, and
Data with 6\TeX

\begin{verbatim}
\cf *KV 488* \edef c\mp{\Comp}\%
\end{verbatim}

is not recommended. The \Comp field contents \texttt{Mozart2} refer to other entries, but the stage \texttt{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{\def \c\m{\Comp}{}\} *KV 488* \%
\end{verbatim}

hence

\begin{verbatim}
\ifcs \c\m{} \is\def{Austria}\%
\end{verbatim}

is true. Of course, more would be needed to make the test work for arbitrary \texttt{Mpieces. Now the} \Comp and all other fields have to exist down to \texttt{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 \texttt{Country} field contents are \texttt{*Austria*}, then after

\begin{verbatim}
\newif\ifTheRightOne
\def \Town{\global \TheRightOnefalse
\ifExistingField \LooksLike \texttt{\Comp}\texttt{\Country}{\texttt{\Austria}*};\%
  \global \TheRightOnetrue
  \fi
}\%
\end{verbatim}

it will be seen that \texttt{\ifTheRightOne} is true.

5.3 Wrapping

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

\begin{verbatim}
\WrapIn \texttt{pseudo field name} \texttt{field name}
\end{verbatim}

is equivalent to

\begin{verbatim}
\gdef \texttt{pseudo field name}{\texttt{field name}}
\end{verbatim}

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

\begin{verbatim}
\def \Town{\WrapIn \c\m{\Country} \texttt{\Comp} \texttt{\Country}{\texttt{\Austria}*}} *KV 488* \%
\ifcs \c\m{} \is\def{Austria}\%
\end{verbatim}

is true, and \texttt{\c\m{\Country}} is available for other purposes.

6 Conversion

Most data base programs are capable of reading and producing ASCII data files. 6\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 \alpha T files. The second converts, for almost every data base program, a \alpha T 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 \alpha T file according to a certain \Type:
\Filter (\alpha T 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. \LaTeX reduces the \alpha T 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 \alpha T file mail.dat with many entries of different \Types sorted on some postal code. If all entries should receive a letter, then \Filtering the \alpha T 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 \alpha T file of Composer. If there is a field TownOfBirth, its contents refer to a \alpha T file of towns, i.e. are of the form *\(a\) town*'*. Every Town has a 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
\IfC\cDntry Is Defined As\{\Austria\%
 {\bf\nAme}, from \nAme, \cDntry\par
\Fi
\Fi
\Filter c:/dat/comp \Type\Composer

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

References

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