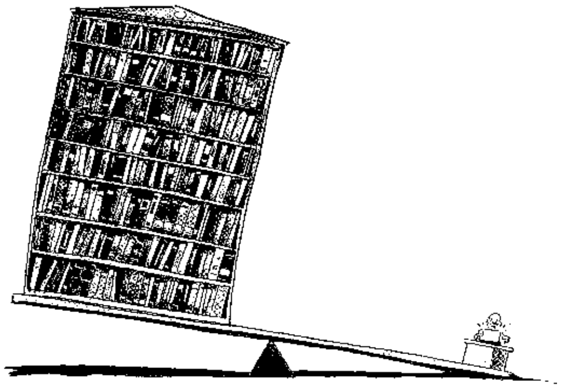
KE Texpress



Load Guide

KE Software Pty Ltd

Contents

Chapter 1

Introduction

Overview

KE Texpress incorporates a general purpose facility for importing data in a wide variety of formats into a KE Texpress database. The importing program interprets a grammar file which defines the format of the data. The only requirement of the data is that it must be in a reasonably consistent format.

The loading facility may also be used to maintain a duplicate copy of a database, possibly on another machine. It may also be used for batch deletions of information.

Chapter 2 describes the data loading facility, called **texload**. This facility may be used to load data directly into a database or to create an intermediate file. A data verification program can be used on the intermediate file for data cleanup. The texload program can then be used to load the intermediate file into the database. Chapter 2 also explains how the texload program is used in setting up duplicate databases, and how an automatic data feed may be loaded.

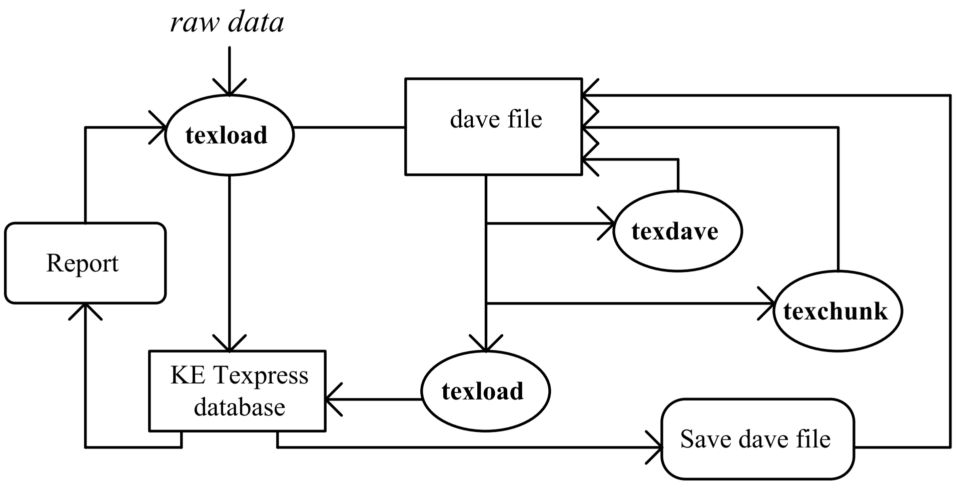
Chapter 3 describes a data cleanup/verification program, **texdave**. Also covered in Chapter 3 is a program for dividing an intermediate data file into more manageable units.

Data Import Flow

A data import flow diagram follows.

The data cleanup/verification program, **texdave** may be used repeatedly on a dave file. Also, **texchunk** can be used to divide any dave file into multiple smaller files.

The **texforms** program provides a command for copying a matching set of records to a dave file.



Data Import Flow

Command Notation

Commands run within KE Texpress can be performed from pull-down menus, or via keyboard accelerators (characters). Often, a command can be performed in several ways. For uniformity and ease of use, only the pull-down menu option is described in the guide.

The *Perform Query* command can be run by selecting the *Query* option on the *Function* pull-down menu in *Query* mode. The guide would describe it as:

```
Perform Query ([Query] Function ⇒ Query)
```

where the description of the command is followed by the pull-down menu option in brackets. The mode is contained in square brackets, e.g. [**Query**] and the pull-down menu name **Function** is followed by ⇒ and the command **Query** exactly as listed on the pull-down menu.

Alternatively, the Perform Query command can be performed from the keyboard, by holding down the **`Control`** key and typing **`Y`**. In the guide, the keystrokes are described as **`Ctrl+Y`**.

Many of the common commands may be invoked directly by a particular function key (provided the terminal supports function keys). Throughout KE Texpress (and the screen images displayed in this document), function keys are represented by the letter `F` followed by the function key number. Thus the command generated by pressing function key number one is represented as `F1`.

Other special symbols used in this document are:

`ESC`	The **escape** key.
↵	The **return** or **enter** key.
`DEL`	The **delete** or **interrupt** key (used to interrupt an operation).
`Space`	The **space bar**.
`Backspace`	The **backspace** key.
`Tab`	The **tab** key.
←	The **left arrow** key.
→	The **right arrow** key.
↑	The **up arrow** key.
↓	The **down arrow** key.

Chapter 2

Data Loading

Overview

KE Texpress's data loading facility, called **texload**, may be used to load data directly into a database or to create an intermediate file which can be manipulated using the KE Texpress data verification tool, **texdave** (refer to Chapter 3). The texload program also allows a duplicate copy of a database to be maintained, and provides a means for data to be loaded from an automatic data feed. A facility also exists for the deletion rather than the insertion of records.

The data to be loaded (called the **raw data**) is generally in a Unix file, but may be generated (or manipulated) by another program, the output of which can be piped into texload using the standard Unix pipe facility. It is also possible to have the raw data being constantly added to a file (called a **feed**). The load program can continually read raw data from the feed and load it into a KE Texpress database.

It is necessary to develop a grammar which describes the format of the raw data. This grammar is stored in a Unix file, and contains information used by texload to identify each of the items/fields for each incoming record.

There are several options which determing the actions performed on each incoming record after it has been parsed according to the grammar description. These actions include:

- checking the record for validity,
- storing the record in the database (after it has been validated),
- updating an existing record with the incoming raw data,
- deleting an existing record, or
- storing the record in a file in dave format (refer to Chapter 3 for more information).

The load program produces an error file which details any erroneous data. Only the DBA may use this program.

Invocation

The KE Texpress program, texload, must be executed from the Unix shell by using a command of the format:

```
     texload [-nnum] [-lpriv] opts dbname
```

where *opts* is one of the following:

```
     [-cehpsu] [-t[file]] -ddatafile -ggrammar
     [-h] -ddatafile -ggrammar -odavefile
     [-cepsuw] [-t[file]] -idavefile
     [-ksu] [-t[file]] -idavefile
```

Arguments appearing in square brackets are optional. The eight loading alternatives are:

- Load data directly into the database. Perform validation till automatic values, calculate expressions and trigger database links.
- Load data into a dave file. Dave format is used to produce, a file which is suitable for texdave (see Chapter 3). Validation is not performed and expressions are not calculated.
- Load data from a dave file into the database. After the data has been repaired using texdave, it can be loaded into the database using this alternative. The data is validated, automatic values filled, expressions calculated and linked databases triggered.
- Load data from a file stored in an extended data file format. This alternative is used to maintain a duplicate copy of a KE Texpress database.
- Loading data directly into a database. Perform validation fills automatic values, calculates expressions reads linked databases. Upon completion of the load the database is not restored to an operational state, but rather is capable of accepting more data without the need to unbitslice.

The available options (those characters listed inside the square brackets) include the following:

`-c`	Perform record validation without actually inserting the records into the database. All expressions are calculated and linked database entries read. The checks performed are identical to those performed when a record is actually inserted. An error file is produced. This option can be used to clean data before records are actually inserted, or to determine if the validation imposed is too restrictive. Duplicate key values are not detected with this option.

`-e` Erase (or delete) records from the database. Deletion is performed using the Key value of the record. The input file need consist only of Key values. For each Key value found in the input file, the corresponding record in the database is deleted.

`-h` Scan the input file using the given grammar. Due to the complexity of some data formats, it may be first necessary to determine the correctness of the grammar. A report is generated indicating the values which would normally be assigned to each field of each record. The report can be prematurely terminated by typing. An example of the use of this option is presented later in this chapter.

`-k` Maintain a duplicate copy of a database. The input file is in extended data file format, containing insertions, edits and deletions. Each input record dictates an action to be performed corresponding to an action performed in the originating database.

`-l`*`priv`*

Set the privilege level at which the records are loaded, to ***priv***. Normally when records are inserted into the database, they are assigned a default insertion privilege level which is the DBA's privilege level. The insertion privilege level, can be changed only to a value between the DBA's privilege level and the lowest privilege of the database.

`-n`*`num`* Estimate that there are approximately ***num*** records to be inserted. This enables texload to select one of two insertion methods. One method is faster for a large number of insertions, while the other is more efficient for a small number. If this option is not specified, then the method selected is that for a large number of insertions.

`-p` Perform a partial linked update. The linked fields in the form are assigned values from the linked database if and only if the linked Key value exists in the linked database. Otherwise they retain any existing values.

`-s` Load the data while keeping the database operational. The database is otherwise closed when data is loaded. To allow the database to remain operational, a slower method of insertion is employed. This is less taxing on the machine's resources, but significantly slower for large amounts of data.

`-t[`*`file`*`]`

Scan the input file Continually for data. After reaching the end of the input file, texload will not complete, but rather periodically monitor the input file for new data and load it as it arrives. This allows a data feed to be set up, whereby incoming data is loaded automatically into a database. The optional argument ***file*** is the name of a file into which data from the input file is copied before being loaded. By specifying different ***file*** arguments more than one texload may load data from the one input file into a database.

`-u` Update the database with the incoming data. Updating is performed using the Key value of the record. If the incoming record contains the same Key value as an existing database record, then the database record is updated with the information in the incoming record. Fields of the incoming record which have a non-null value replace the corresponding fields in the database record.

`-w` Completely replace the existing record with incoming record including replacing existing fields with null incoming fields.

Options requiring a file name are:

`-ddatafile`

Read the raw data from the Unix file, ***datafile***. If the file name given is "**-**" then the standard input is read allowing other programs to pipe information into texload.

`-ggrammar`

Read the grammar (the description of the format of the raw data) from the Unix file, ***grammar***.

`-odavefile`

Load the data into the data format file, **davefile**. The records are appended to the file.

`-idavefile`

Load the data from the data format file, davefile, into the database.

During the loading of data, it is essential that there are no active database users. To enforce this, texload closes the database before commencing the load (unless the -s option is used). If, however, there are people using the database the following error message is produced:

```
There are n people using the database.
```

and the load is aborted.

Grammar File

The grammar file is used to describe the format of the raw data. It defines the delimiters used in the data, and the order in which fields occur. This section contains a full description of the grammar file, indicating the various data input formats which can be recognised.

As the grammar file describes the order in which the fields of each record appear in the data file, it is necessary to identify uniquely each field of the Insertion form of the database. The field Id assigned within the forms editor is used for this purpose (refer to the KE Texpress Design Reference Manual).

Delimiters

Delimiters are character sequences which separate fields in the data. They may consist of a series of control characters, or may be a set of printable characters which do not appear in the field. For example, consider the following data string:

```
John Smith,124 Waldin Valley Rd,Melbourne
```

Here the character '**,**' is used as the delimiter to separate the fields. This is simple example of the data stream expected. The texload program is not limited to single character delimiters, but can handle multiple character delimiters and multiple pattern delimiters (i.e. delimiters that may be written in more than one way).

Leading and trailing spaces are removed from the data before it is assigned to a field. Hence it does not matter whether spaces surround delimiters. The above data is treated in the same manner as the data string:

```
John Smith  ,  124 Waldin Valley Rd   ,   Melbourne
```

Grammar Format

The format of the grammar file follows the basic pattern:

```
delimiters fieldid delimiters ;
```

Each field Id must have leading and/or trailing delimiters. It is not necessary to have both. The character '**;**' is used to separate statements of the grammar. Comment lines are permitted and are introduced with the # character. The permissible delimiter formats are:

- ["delimiter string"]
- ["delimiter string" -> fieldid]

- {"delimiter string"}
- {"delimiter string" -> fieldid}

For example, if a name is to be inserted into the field, *name*, and two telephone numbers into the fields, *phone_1* and *phone_2*, where the input data is:

```
John Smith : 890-1234 , 890-1235
```

then the following grammar file can be used:

```
name [":"] ; phone_1 [","] ; phone_2 ["\n"] ;
```

In the grammar file, spaces and newlines (white space) are insignificant. Thus:

```
name [":"] ;
phone_1 [","] ;
phone_2 ["\n"] ;
```

produces the same result as the previous grammar file.

In the above grammar, the first field Id is *name*. (This is actually an item Id but defaults to the first field of that item.) This is the name of the field in the Insertion form into which the first data field is to be placed. Notice that no leading delimiters are specified. Following this is the delimiter specification. The character, '**:**', is the delimiting string. A semicolon is used to separate the statements. Following this is the next field Id, *phone_1* (i.e. the first field in the phone item). The delimiting string for this field is '**,**'. Finally the last field Id, *phone_2*, is given followed by the delimiter, '**\n**', which is used to represent the newline character. This states that the second field of the phone number is terminated by a newline. Thus each record starts on a new line.

An acceptable raw data sequence for this grammar is:

```
John Smith : 890-1234 , 890-1235
Bill Smith: 123-4567, 123-4568
Jack Jones :98-7654 , 456-7890
Fred Flintstone :,
```

Upon scanning the grammar file, texload places all data up to the delimiter, ':', into the field called *name*. If a colon occurs in the data, this is taken as the delimiter, so it is important to ensure that the specified delimiter can never appear in the data for that field.

For example, if the following data is found:

```
Bill :Smith : 890-1234 , 198-1235
```

then the string, **Bill**, would be placed into the field, *name*, and the string, **Smith : 890-1234**, into the field, *phone_1*.

For each field, texload reads only as many characters as can be accepted by the field. It then discards all subsequent characters up to the delimiter. If a field is truncated, then an error message is produced in the file. When the *name* field is complete, texload proceeds to the *phone_1* field, and so on until the end of the grammar is reached, at which point it is assumed that the record is complete. The record is validated and, if valid, inserted into the database. The next record is then commenced, restarting at the first line of the grammar.

Consider the following data:

```
1:Fred
2:Fred's House
3:3098
1:Bill
2:Bill's House
3:3212
```

In this example leading delimiters are used. A suitable grammar is:

```
["1:"] name ;
["2:"] address ;
["3:"] postcode ;
```

A more complete specification using both leading and trailing delimiters would be:

```
["1:"] name ["\n"] ;
["2:"] address ["\n"] ;
["3:"] postcode ["\n"] ;
```

For multiple character delimiters, every character in the delimiting string must be present before the field is recognized.

Consider the grammar:

```
name [";:;"] ;
phone_1 ["..."] ;
phone_2 ["--\n"] ;
```

This indicates that the delimiter for *name* is the string, ";:;". Thus for the input data:

```
Bill ;: John ; Smith ;:; 890-;:;0 ... 890--000--
John Smith ;:; ... 123-4567--
```

the following records are generated:

	Record 1	Record 2
name	Bill ;: John ; Smith	John Smith
phone_1	890-;:;0	
phone_2	890--000	123-4567

The *phone_1* field of the second record is empty since no data appears between the two delimiters. If texload is invoked with the **-h** option the following report is generated:

```
Field name
        "Bill ;: John ; Smith"
Field phone_1
        "890-;:;0"
Field phone_2
        "890--000"
End of record

Field name
        "John Smith"
Field phone_2
        "123-4567"
End of record
```

Extended Grammars

It may be necessary to use more than one delimiter for the same field. For example, consider the following data file:

```
fred ; 890-1234 , 123-4567
bill : 456-1234 , 123-7654
```

where the first field may be delimited by either the character, '**;**', or the character, '**:**'. A grammar for this data could be:

```
name [";"] [":"] ;
phone_1 [","] ;
phone_2 ["\n"] ;
```

Notice that the two possible delimiters for the field, *name*, are both specified. This is the format for all multiple pattern delimiters.

It may be necessary to include control characters in a grammar file. One example of this is the character, '**\n**', which is used to denote a newline. The following table contains the special characters which are recognised by texload. In general, for all letters not in this table, the sequence, '**\a**' (where 'a' can be any letter), simply represents the letter, '**a**'.

Symbol	Character
`\b`	Backspace
`\n`	Linefeed (new line)
`\t`	Tab
`\f`	Formfeed (new page)
`\r`	Carriage return
`\\`	Backslash character "\"

Another way of specifying special characters is to use their octal value in the **ascii** character set. The sequence, '***nnn***', can be used, where ***nnn*** is the octal value of the character required. For example, a newline can be specified as either '**\n**' or '**\012**' where 012 is the octal value of the newline character.

Delimiters may also contain the standard pattern matching characters used throughout KE Texpress (refer to the KE Texpress User Guide). These characters are as follows:

`^` *(circumflex)*

If this character is at the start of the delimiter, then the delimiter must appear at the start of a line in the raw data. If this character appears anywhere other than at the start of the delimiter, it is taken as a literal character which must be matched within the data.

`$` *(dollar)*

If this character is at the end of the delimiter, then the delimiter must appear at the end of a line in the raw data. If this character appears anywhere other than at the end of the delimiter, it is taken as a literal character which must be matched within the data.

`?` *(question mark)*

This character matches any single character at the corresponding position in the data.

`*` *(asterisk)*

This character matches zero or more characters (including zero) at the corresponding position in the data.

`[str]` *(character range)*

This sequence implies that the next character of the data must be matched by one of the characters in the sequence, ***str***. The sequence, ***str***, may consist of individual characters and ranges of characters (for example, the range, **a-m**, indicates all of the characters in the first half of the alphabet. If the first character of the sequence is '^' then this implies that the next character of the data must **not** be matched by one of the characters in the remainder of the sequence.

`{str}` *(repeating character range)*

This sequence can be used to match zero or more characters of the data which are contained within the sequence, ***str***. This sequence is interpreted as described above.

`x` *(where 'x' is any other character)*

This character must be matched by the same character at the corresponding position in the data.

\ *(literal next)*

This character can be used to remove the special effect, if any, of the next character in the delimiter.

Synchronizing Grammars

It may be desirable in a grammar to be able to select certain unique delimiters which can be used as synchronization points. This is useful for skipping fields in the data input or for error recovery on inconsistent data. Consider the following grammar:

```
name [","] ;
address [","] ;
suburb [":"] ;
phone_1 [","] ;
phone_2 ["\n"] ;
```

It can be seen that whenever a newline character is found in the data file, the end of the current record has been reached. Now consider the following data input:

```
Bill Smith , Melbourne : 123-4567
```

This data is not in the correct format and hence is not acceptable to texload. However, the two unique delimiters, '**:**' and '**\n**', allow for a sensible interpretation of the data as shown in the following table:

Field	Data
name	`Bill Smith`
address	
suburb	`Melbourne`
phone_1	
phone_2	`123-4567`

To achieve such an interpretation, it is necessary to direct texload to search for the next delimiter as well as the synchronizing delimiters, '**:**' and '**\n**'. To specify this, it is necessary to place the synchronizing delimiters between the characters, '**{**' and '**}**'. Thus the grammar could be specified as:

```
name [","] ;
address [","] ;
suburb {":"} ;
phone_1 [","] ;
phone_2 {"\n"} ;
```

The data is placed into the field which the delimiter matches. This explains why the phone number in the previous example is placed in the *phone_2* field and not the *phone_1* field.

It is good practice to place the **{"\n"}** delimiter at the end of a grammar if the raw data records are provided one per line. This results in the current record being terminated whenever a newline is encountered.

Reserved Field Ids

Grammars can make use of several special reserved field Ids. The reserved field Ids include:

`null`	Data assigned to this field Id is discarded. There is no restriction on the number of times this field Id can be used.
`comment`	Data assigned to this field Id is placed in the special comments of the record.
`end`	This field Id can be used only in a directed grammar (after the -> symbol, see the next section). It indicates that the end of the record has been reached.
`all`	This is a short-hand way of indicating all field Ids.

Directed Grammars

Consider the following data format:

```
NM      Bill Jones
PS      Programmer
PH      60 3217 Ext 24
##
NM      Mary Jones
PH      472 1353
##
```

In this example, the delimiter, **NM**, represents the *name* field, the delimiter, **PS**, represents the *position* field, and the delimiter, **PH**, represents the *homeph* field if no position is specified or the *busph* field otherwise. Following the phone number is a terminator which is to be discarded. It is necessary for a single delimiter to select one of two fields depending on other data in the record. One way to parse this data correctly is to use the following grammar:

```
["^NM "]        name ["\n"] ;
["^PO "]position ["\nPH " -> busph] ;
{"^PH "}homeph ["\n"] ;
["^PH "]busph ["\n"] ;
{"^##"} null ;
```

The symbol, **->**, indicates that, after assigning a value to the *position* field, texload should continue at the *busph* field. The use of the characters, **{** and **}**, to surround the first occurrence of the delimiter, **PH**, enables the *homeph* field to be loaded if no *position* field appears in the record.

Automatic Field Incrementing

If, in a grammar, an item Id is specified instead of a field Id (i.e. no underscore and field number is appended to the field Id), the data is placed in the first empty field within that item. If, however, a field Id is specified, the data is always placed in that particular field.

Consider the following raw data sequence:

```
AD 23 Hill Street
AD South Hill
AD 3412
```

A grammar to place each line of the data in a different field of the address item is:

```
{"^AD?"}        address_1 ;
["^AD?"]address_2 ;
["^AD?"]address_3 ;
```

A simpler grammar using the automatic field incrementing facility is:

```
{"^AD?" -> address} address ;
```

If data is assigned to an already full item, the record is considered to be in error and is not inserted. A copy of the record is placed in the error file.

Grammar Options

Several grammar options are available. These options are used to alter the way a record is loaded. These option entries must be placed at the end of the grammar file. The reserved field Id, **all**, may be used if all fields are to be selected by the option. The options are as follows:

`:noscan` *`id1 id2`* `. . . . ;`

Disallow scanning for synchronizing delimiters. Thus for the specified field Ids, texload does not search for delimiters enclosed in { and }. The only delimiter used is that of the current field Id.

`:back` *`id1 id2`* `. . . . ;`

Allow scanning for the current record to continue from a point earlier in the grammar. Normally, it is assumed that a record is complete whenever the last field of the grammar is reached or if it is necessary to scan backwards through the grammar. This option indicates that a new record should not be commenced when texload is forced to scan backwards through the grammar for the given field Ids.

`:join` *`fieldid`* `[("`*`string`*`")] . . . . ;`

Join field information (the square brackets indicate that ("*string*") is optional). By default, if the same field Id is used more than once in the grammar, texload overwrites the contents of the field each time that data is assigned. This option can be used to join (append) the new data to the data already in the field. It is possible to specify an optional joining string to be placed between the field contents and the new data. If a joining string is not specified, the space character is used. If multi-field items are specified, the data is spread over all of the fields. Where possible, data is broken only at word boundaries.

`:replace` *`id1 id2 . . . . ;`*

Used in conjunction with the data update option of texload to indicate that existing values of the given field Ids should be replaced by the incoming value even if the incoming value is null.

`:exact` *`id1 id2 . . . . ;`*

Maintain leading and trailing spaces in the data for assignment to the field.

Blank Padded Grammars

Incorporated into texload is a facility to load blank padded data. This is data for which the fields are in a consistent order and the raw input data is padded to a known and constant length. For a blank padded field specification of *n* characters, the raw data for the field must contain exactly *n* characters. This implies that each record is a fixed size.

A blank padded data grammar consists of lines of the form:

```
fieldid  (n);
```

where ***fieldid*** is the field Id into which the data is to be loaded, and ***n*** is the number of characters associated with the field in the raw data. The sum of all of the lengths in the grammar should equal the record length. The **join** option is valid for blank padded grammars, and the reserved field Ids, **null** and **comment**, may also be used.

As an example, consider the following data:

```
John Citizen    :John's House   :859-6244       :
Bill Person     :Bill's Home    :89-6897:
```

A suitable grammar is:

```
name (13)
null (1)
address (13)
null (1)
phone (9)
null (2)
```

Note the use of the **null** field Id, to ignore the colons and newline from the raw input data.

Database Insertions

Before each record can be inserted into the database, a number of automatic operations are performed. These operations are not performed when loading data into a data format file for use with texdare.

If the Insertion form contains a Key item, this item must contain a value, and the value must not be associated with any record already in the database. If this condition is not satisfied, the erroneous record is not inserted into the database, but rather, is copied into an error file.

Following this, all automatic items are filled. Then, if the Insertion form contains an item which represents the Key of a linked database and this item contains a value, all related fields have their values filled from the appropriate record in the linked database. After this, all expressions are calculated.

The final operation performed is record validation. This involves checking all fields that have validation expressions, and checking all hierarchies for completeness. Type checking is also carried out, to ensure that illegal characters are not found in the data. If any errors are found, the record is not inserted, but rather an error message is written to the error file and, if possible, the record is temporarily inserted. A list of the possible error messages and their meaning is provided in Appendix B.

Database Updates

When records within the database are being updated, a set of operations similar to those performed by data insertion are performed. To be able to use the texload facility for updating database records, the database must contain a Key item, and the input data must supply a Key value for each record.

If the Key value of the record to be updated does not exist in the database then a new record is created with that Key value. Otherwise the old database record is retrieved and the record is updated with the new data. The type of update required is controlled by the options given when texload is invoked, and by any options set in the grammar file. By default, texload updates only fields for which data is supplied in the input file. By using the record replace option (**-w** option) the complete record is replaced with new values. Any fields which do not have data supplied in the input file are. The grammar option **:replace** allows certain fields to be replaced always with new data, even if this means setting to empty a field which previously held data.

Database Deletions

By using the texload option, -e, it is possible to delete records from a database in a batch manner. To delete records the database must contain a Key item, and each record in the input file must specify the Key value to be deleted. Any other data found in the input file is ignored.

If the Key value supplied in the input file does not exist in the database, an error message is placed in the error file and processing is continued. When the database record is deleted, it is removed permanently from the database, along with any comments or history versions of the record.

Interactive Loading

Generally when texload is modifying a database, the database is closed. This allows certain optimizations to be made during the load. Whilst the database is closed, access to the data is temporarily denied. Upon completion of the load, the database is opened automatically.

A method is provided whereby the data may be loaded without the database being closed (**-s** option to load). By keeping the database each interactive open, texload competes for access to the data in the same way as user, and hence data loading optimizations are not performed. This results in a significantly slower loading rate.

An interactive texload does not save any error messages in an error file but rather prints each error message to standard error (stderr). To save a copy of any error messages in a Unix file, it is necessary to re-direct standard error.

Continuous Loading

Some applications require the continuous loading of data into a database. A facility is provided (**-t** option) which allows for the continuous loading of data from a file. This option instructs texload to monitor a file continually for data to be loaded. Once some data is found, a copy of the input file is made and the data in the copy is loaded into the database. The input file is set to empty, ready to accept more.

The name of a Unix file may be optionally supplied to the continuous load option. This name will be used as the file into which to copy the input data before loading it into the database. By default the file used is the name of the input file with **.t** appended to the end. By supplying the optional name it is possible to have more than one texload feeding data into the database. Each texload running requires its own file into which it can copy the input file.

To ensure that the input file is accessed in a consistent manner, a lock file is used. The name of this file is the same as the input file with **.l** appended to the end. It is important that any programs which write data into the input file create this lock before writing to the file and remove it once the write is complete.

Duplicate Databases

By setting a database option and using the duplicate database option (**-k**) of texload it is possible to have a duplicate copy of a database maintained. The database option required is **duppath** which is set to the name of the file into which extended data file format records are written. These records interpreted by texload and the necessary updates to the duplicate database are performed.

To set up a duplicate database, a copy of the original database is first required (via the texcopy command). The database option duppath should be set to a particular file. Then, texload should be started up with the duplicate database, continuous loading and interactive loading options (i.e. -kst) and the name of the input file set to the same file as duppath. It is important to ensure the texload is running continuously if an exact duplicate is to be kept. To ensure this, it is recommended that the texload process is started by the Unix system startup file (e.g. /etc/rc).

Monitoring Data Loading

The progress of texload can be monitored from the **Rebuild/load/progress** command on the Administrator Maintenance menu. The report procedure is similar to the Automatic data rebuild report method (refer to the KE Texpress Design Guide). If the interactive option of texload is being used, a report is not available, rather all error messages and report details are printed to standard output.

Generally texload is quite efficient at loading large amounts of data**.** This efficiency is due to optimizations which texload performs based on the fact that the database is closed. One of these optimizations is to modify the index to accomdate. The operation which takes the segment index file and converts it into an optimized form is known as **unbitslicing**. Once the load is complete the segment index file needs to be returned to its standard form. This operation is known as **bitslicing**.

When texload is run it first performs an **unbitslicing** of the segment index file, then loads the data, and finally performs a **bitslicing** of the descriptor file.

System Failure

When manipulating large numbers of records, texload can make many record insertion optimizations. However, these optimizations introduce brief critical periods during which recovery of the database after an operating system failure may not be possible If operating system failure occurs during one of these critical periods, the database must be restored from backup. For this reason, it is essential that the database is backed up before a texload is attempted. If the database has not been backed up, the message:

```
Database must be backed up before starting load
```

is displayed, and the backup must be performed before texload can commence.

The texload program is capable of recovering from most operating system failures (provided that they do not occur during one of the critical periods described above). A texload may be resumed after an operating system failure by viewing the progress report. The report will display the message:

```
Do you wish to recover?
```

If an affirmative reply is given then texload will attempt to continue from where it was before the system crash. The message:

```
Recovering ...
```

is displayed. If recovery is not possible, the message:

```
Can't recover. Restore database from backup
```

is displayed. The database must be restored from backup and the texload restarted.

If a negative answer is given the message:

```
Can't recover. Restore database from backup, then
restart load
```

is shown to indicate that that the database needs to be restored before it is accessible again. Can't recover. Restore database from backup

Memory Requirements

The texload program requires a considerable amount of memory to perform efficiently. In fact, the more memory it can use, the faster the loading of data can be performed. By default, texload uses (actually set upon installation) 1mb but this value can be increased by the database if the machine has sufficient resources. This can be achieved using the database option, **loadmemory**. Refer to the KE Texpress Design Guide for a description of database options.

Chapter 3

Data Verification

Overview

To allow users to maximise the integrity and accuracy of data to be loaded into a database, KE Texpress provides a tool for the verification of data. This tool is known as **texdave** and can be used to check and correct data types, sets and ranges of values, data integrity and exceptions to data patterns.

Data Preparation

As described in the previous chapter, the program, **texload**, can be used to convert raw data into a format which can be manipulated by **texdave**. This produces a file containing each loaded record in a format referred to as the standard data format. Similarly, at the completion of the data verification, the standard data format file can be loaded into the database, again using **texload**.

As the number of records in the intermediate file may be large and hence unmanageable, KE Texpress provides another utility, **texchunk**, which divides a file into a collection of smaller files.

Dividing a KE Texpress Data File

The program, **texchunk**, is used to divide a standard data format file into smaller more manageable files. It is invoked from the Unix shell by a command of the form:

```
texchunk [-nnum] [-pstr] [-snum] dbname filename
```

where ***dbname*** is the database name and ***filename*** is the name of the data form file to be divided. The optional prefix argument **-p** is used to indicate that files are to be named with the prefix, ***str***. By default a prefix of **xxx** is used. Files are created with names in the format:

```
prefix.s
```

where ***s*** is an incrementing numeric suffix, commencing at 1. The optional suffix argument **-s** is used to indicate that the suffix should commence at the numeric value, ***num***.

The optional argument **-n** is used to specify the number of records to be loaded into each file. By default, the data format file is divided into blocks of 1000 records. The original file is not altered.

For example, the command necessary to divide a standard data format file for the restaurant database, called **data**, into blocks of 500 records, and place these blocks in files commencing with the name, **dave.1**, would be as follows:

```
texchunk -n500 -pdave restaurant data
```

As each new file is created, a one line message is displayed indicating the number of records in the file. After completion an indication of the total number of records is also displayed.

It is generally not possible to re-combine files divided using the **texchunk** program.

Invocation

Once the data has been prepared and divided into manageable sections, the data verification program can be executed from the Unix shell by typing a command of the form:

```
texdave dbname datafile
```

where *dbname* is the name of the database. This is used to verify and repair all of the records in the file, *datafile*. The user must specify the full path of *datafile* if it is not in the user's current directory.

If the data is not to be divided via **texchunk** prior to running **texdave**, it is strongly recommended that the data file first be copied and **texdave** run on the copy, as a precaution. For example:

```
cd dbpath                    (where dbpath is the path of the database)
cp data data.dave            (where data.dave is any file name)
texdave dbname data.dave
```

Only database users at privilege level 0 are permitted to run this program.

Data Verification Procedure

The steps involved in the verification of data are as follow:

(1) All or a subset of the records in the data file are selected. This is referred to as performing a query and is achieved using the same facility as that used in (KE Texpress User Guide).

(2) The selected records are distributed into bins according to the value of their secondary key. The secondary key can be a field or an item from the Insertion form, and is selected using the standard KE Texpress field/item selection process.

(3) A bin is selected. Display mode is entered. From here the records can be viewed, copied, altered both individually or globally, and so on.

(4) At the termination of Display mode, either another bin is selected or another query commenced.

(5) At the completion of data verification, all of the changes can be saved to the original standard data format file, thus becoming permanent. If exit is selected without a preceding save operation, the changes will be discarded.

Performing a Query

To retrieve records from the data file, a **texdave** query uses the same facilities within **texforms** (refer to the KE Texpress User Guide). Thus the Query form is displayed and the user can enter query criteria into the Query form fields. As in **texforms**, patterns and boolean operations are supported. There is no index for the dave file, however, so a **texdave** query involves an exhaustive search of the file data, which depending on the size of the data file, can be quite slow.

Once the required field values have been entered, the Query command (`[Query] Function ⇒ Query`) will call up the secondary key selection form.

If no query field values had been entered, all of the records in the data file are retrieved. Otherwise, each record is tested to determine whether it contains all of the query terms entered. Only those which match these terms are retrieved from the data file.

For a complete description of the query facilities refer to the KE Texpress User Guide.

The user can then nominate a field or item as the secondary key, which will be used to distribute the retrieved records (from the query) into bins. (See the next section Secondary Key Selection for more details). Only after the secondary key has been selected, or the user chooses not to nominate a secondary key, does the query commence.

Secondary Key Selection

Once the query field values have been entered, a field or item from the Insertion form may be specified as the secondary key. Each record matching the query is placed into a bin determined by the value of this secondary key. If the secondary key is a field, its value is the field's contents. If the secondary key is an item, its value is the concatenation of the contents of its fields.

The Insertion form is displayed with each field of the form represented as an unbroken sequence of underscores. The commands available are as follows:

Select item/field ([Secondary key] Function ⇒ Select item/field)

Choose the current field or item as the secondary key.

Exit ([Secondary key] Function ⇒ Exit)

Exit without choosing a secondary key. All records matching the query are placed in the one bin.

Next item/field ([Secondary key] Move ⇒ Forward an item/field)

Move forward to the next item/field on the Insertion form.

Previous item/field ([Secondary key] Move ⇒ Backward an item/field)

Move backward to the previous item/field on the Insertion form.

Next an item ([Secondary key] Move ⇒ Forward an item)

Move forward to the next item on the Insertion form.

Previous an item ([Secondary key] Move ⇒ Backward an item)

Move backward to the previous item on the Insertion form.

First item ([Secondary key] Move ⇒ Start of form)

Move to the first item (beginning) on the Insertion form.

Last item ([Secondary key] Move ⇒ End of form)

Move to the last item (end) on the Insertion form.

Find an item ([Secondary key] Move ⇒ Find an item)

Find an item on the Insertion form by entering the leading letters of its prompt and then move to that item.

If a secondary key is selected, a message of the form:

```
Exact character or Alphabetic character matching?
```

or

```
Exact character, Alphabetic character or Numeric matching?
```

appears, depending on whether the data for the selected object can be compared numerically or not. These options determine the type of matching to be used in the distribution of records into bins. They are as follows:

Exact

Use exact string matching. Thus two records are put in the same bin if and only if the strings associated with their secondary keys are exactly equivalent.

Alphabetic

Use alphabetic (or text) matching. Thus two records are put in the same bin if, when broken into words and then converted to lower case, the strings associated with their secondary keys are equivalent.

Numeric

Use numeric matching. A number of values can be entered, to define the lower and upper bounds of each bin. A record is placed in a bin if the numeric equivalent of the string associated with the secondary key is greater than that bin's lower bound and less than or equal to its upper bound. This option is available only when the secondary key is a numeric field or an item with a single numeric field or a KE Texpress library item which can be ranged.

Numeric Secondary key - Ranges

If the numeric secondary key matching option is selected, then the screen is cleared, and a message appears at the top of the screen. :

```
Ranges for matching bins
```

The cursor is positioned near the top of the screen and at the left hand edge. Up to 60 ranges can then be entered. The range values are automatically maintained in numerically increasing order, and, at the completion of each range, may be automatically repositioned on the screen.

Each range value can be entered (and subsequently modified) using Text edit mode commands. Additional commands available include:

Query ([Display] Function ⇒ Query)

Terminate the entry of ranges and perform the query. The records which match the query terms are divided into bins defined by the numeric ranges entered. Two extra ranges are created to hold the records with secondary key value from -Infinity to the lowest range entered, and from the highest range entered to +Infinity.

Forward ([Display] Move ⇒ Forward)

Move forward to the next range value. As long as fewer than 60 range values have been entered, a new value can be entered in the blank range immediately after the last non blank range. The range after this blank one is the first range value.

Backward ([Display] Move ⇒ Backward)

Move back to the previous range. The range before the first is the (blank) range after the last.

Bin Summary

Once the user has specified a distribution type, and range if relevant, the query is performed. During this time the following message is displayed:

```
Searching dave file ...
```

Each record in the data file is tested against the specified query terms in the usual way. The matching records are placed into bins according to their secondary key value and the type of secondary key matching employed.

If no records in the data file match the query, the following message is displayed:

```
No matching records
```

Typing any key enables the user to commence the next query.

On completion of the query, if only one bin is created by secondary key matching, then Display mode is automatically commenced. Otherwise, a summary of bins is displayed, with a one-line entry for each bin with at least one record. This entry is called a bin description, in the form:

```
Bin         Count        Secondary key
```

where **Bin** is a unique number used to identify the bin, **Count** represents the number of records in that bin and **Secondary key** is the secondary key value for all records in the bin. This secondary key value may be truncated at the right edge of the terminal screen. In the top, right hand corner of the screen is a message of the form:

```
(x bins, y matches)
```

where ***x*** represents the total number of bins created by the query and represents the number of records in the dave file which match the query.

If the numeric secondary key matching option is selected then the secondary key section of the bin summary is replaced by two numeric quantities, the lower bound and the upper bound of values in the bin. Such a bin contains values greater than the lower bound and less than or equal to the upper bound.

The bin summary is the central point from which all data verification and repair can occur. The summary itself is intended to indicate errors and inconsistencies in the values associated with the secondary key. Display mode, similar to that provided within User (refer to the KE Texpress User Guide), can be employed on the records in each bin to view, copy and correct records both individually and collectively.

The following commands are available while the bin summary is displayed:

Next Screen ([Bin summary] Function ⇒ Forward)

Display the next screenful of bin descriptions. On the last screenful of bin descriptions this command recommences the bin summary at the first screenful. If the bin summary is less than one screenful in length, this command is disabled.

Select a bin ([Bin summary] Function⇒ Select a bin)

You are prompted to enter a bin number and Display mode commences on the records in that bin.

Print/Copy summary ([Bin summary] Function⇒ Copy summary)

Print a copy of the bin summary or save it in a file. Use the TAB key to move between the printer list and the file list. To save in an existing file, move the cursor to the upper box in the file list (using TAB), to the file, and press ↵ to select it. If saving in a new file, enter the new file name in the lower box of the file list. To exit without selecting a file or printer enter F1 or Ctrl+X.

Merge bins ([Bin summary] Function ⇒ Merge bins)

Merge two bins together. When selected, the prompt:

```
Enter first bin number:
```

is displayed and a bin number can be entered. Following this, the prompt:

```
Enter second bin number (first bin = n):
```

appears, where ***n*** is the first bin number entered. Again a bin number can be entered. The records in the two bins are merged into one. If the strings associated with the bin descriptions of the two bins are different, then the string associated with the resultant bin is set to:

```
- MERGED -
```

Otherwise the same string is retained. The two original bins are removed and the new bin is placed in its correct position in the bin summary.

If no bin number is entered in response to one of the above requests, this command is aborted. Invalid bin numbers are rejected.

Exit ([Bin summary] Function ⇒ Exit)

Terminate the display of bin descriptions and commence the next query.

If only one bin is created by secondary key matching, then the bin summary display and selection process is omitted, and Display mode is automatically commenced.

Display Mode

When Display mode is entered, the bin summary is cleared from the screen and replaced by the Display form. This form is either the Insertion form or a Report form (refer to the KE Texpress User Guide for information on designing Report forms). Changing the Display form is described in the next section.

Shown in the Display form is information associated with the first record(s) in the selected bin. If a Report form is used and this form is less than half the screen in size, then User displays as many records as possible on the screen at one time.

In the right-hand corner of the menu bar, is a message:

```
  Record x of z
```
or
```
  Records x to y of z
```

where ***z*** represents the total number of records in the bin and ***x*** (or ***x* to *y***) indicates which of those records is currently being viewed. It is initially 1. The mode indicator is set to **Display**.

If the Display form is larger than the screen, then the screen acts as a window which can be positioned to show any portion of the form. Initially, the top left hand corner of the form is displayed. If the form is larger in one dimension but less than half the screen in the other, then multiple records are still displayed and the scrolling of the window occurs in the larger dimension only.

Many Display mode commands operate upon a single record only. If such a command is invoked and more than one record is displayed on the screen, it will operate on the current (highlighted) record. The commands, **`[Display] Function ⇒ Forward a record`** and **`[Display] Function ⇒ Backward a record`** are used to move forward to the next and backward to the previous record respectively. The order of the records on the screen is from left to right and then from top to bottom. The record after the last on the screen is the first.

For the remainder of this chapter, it is assumed that only one record is displayed on the screen. If this is not the case, then the above selection procedure must be performed first to identify the record .

The full list of Display mode commands is described in the following sections. It is possible to precede the Display mode commands with a number. This number is referred to as the **repeat count** and indicates the number of times the command should be repeated. This count is applicable only to Display mode commands for which it makes some sense to repeat. For all other commands, it is simply ignored.

Function Menu

Display Bin Summary ([Display] Function ⇒ Bin summary)

Display the bin description for the current bin. A message of the form:

```
Bin no - string  - Count = n
```

is displayed where ***no*** represents the number of the current bin, ***n*** indicates the number of records in the bin and ***string*** is the secondary key value. This value may be truncated if there is insufficient space.

If the numeric secondary key matching option was initially selected, then the ***string*** value in the above description is replaced by a sequence of the form:

```
lower < X <= upper
```

where ***lower*** and ***upper*** represent the lower and upper bounds of the bin, respectively.

Change Display form ([Display] Function ⇒ Select Display form)

Select a new Report form for display. The message:

```
Enter Display form name:
```

appears, and the name of the Report form can be entered. This Report form is then used as the Display form, and the screen is altered to reflect this, possibly changing the number of records displayed on the screen. This form remains the Display form until changed or until the end of this session.

If no name is given, the Insertion form is used for displaying records. In this case, only one record is displayed at any time regardless of the dimensions of the Insertion form.

If a Report form containing a sort order is selected for displaying the matching records, then this ordering information is ignored.

Show comments ([Display] Function ⇒ Show comments)

Display the comments associated with this record. This is applicable only if the record has comments associated with it. The record description is cleared from the screen, and the associated comments are displayed using the Text perusal facility.

Exit ([Display] Function ⇒ Exit)

This command terminates Display mode and returns the user to the previous screen, either to the bin summary, if only one bin was created by the initial query, to the Query form to commence the next query.

When the operation being performed is one which reduces the number of records in a bin to zero (like Delete or Discard) then an implied Exit command is performed, and Display mode is terminated.

Move Menu

Next record ([Display] Move ⇒ Forward)

Move forward through the records in the bin to the next record. The list is effectively circular, so the record after the last is the first. If more than one record is displayed on the screen, then the next screen of records in the bin is displayed.

Previous record ([Display] Move ⇒ Backward)

Move backward to the previous record (or screen of records) in the bin. The record before the first is the last.

Screen Up ([Display] Move ⇒Up)

Move the form window half a screen up, if the form size is larger than the screen.

Screen Down ([Display] Move ⇒Down)

Move the form window half a screen down, if the form size is larger than the screen.

Screen Left ([Display] Move ⇒Left)

Move the form window half a screen to the left, if the form size is larger than the If it is already at the left hand boundary, then the terminal bell sounds and the command is ignored. The window stays at this position until moved, or until termination of Display mode.

Screen Right ([Display] Move ⇒Right)

Move the form window half a screen to the right, if the form size is larger than the screen.

Edit Menu

The edit commands on this menu make changes to the data file. Once these changes are saved, these changes became permanent. When any edit command is terminated, the user is returned to Display mode on the record. The edit commands are:

Edit record ([Display] Edit ⇒ Edit record)

Edit the record. The user enters Edit mode with access to all the editing facilities available in **texforms.** (refer to the KE Texpress User Guide). The Insertion form is used for editing with the mode indicator set to Edit.

Throughout **texdave**, all operations relating to key items are disabled. The Edit mode features which are supported include the default value load command, expression calculations, performed whenever an inter-dependent field value is altered, and record validation. The comments can also be edited, exactly as in **texforms**.

Termination of record editing can be performed in two ways. Exiting ([Edit]Function ⇒ Exit) will avoid updating the record. Then, if the information in the Insertion form has been modified, **texdave** prompts for confirmation:

```
Exit from Edit mode? [yes] [no]
```

If yes is chosen, the alterations are discarded and the record remains unchanged. If no is chosen, Edit mode resumes.

Saving ([Edit]Function ⇒ Save) replaces the old version of the record with the modified one. The record is not validated before it is saved. The user can manually perform validation checks on the record at any time ([Edit] Info ⇒ Valaidate data)

A special case of editing occurs when only the Comments are altered, as this has no effect on the data in the record. So in this case no re-insertion of data is performed, but instead the new Comments are stored and linked to the current record.

Edit all matching records ([Display] Edit ⇒ Edit matching records)

Edit all records in the current bin. A blank Insertion form will be called up, into which the user enters information. On completion of editing, for each field in the form containing a value, the corresponding field in every record in the bin is replaced by that value.

On selecting this option the user is in Edit mode with access to all the standard editing facilities of **texforms**. However, all the additional commands offered for insertion are disabled, except for the default value load command. The mode indicator is set to Global.

There is an additional editing command provided. As described above, only fields which contain a value have their values substituted into the records in the bin. This precludes the possibility of setting the value of a particular field to be empty. To circumvent this problem, the Wipe Field Clear command is provided. This command can be used to alter the state of the Field Clear setting for a given field. When the global substitution occurs, in each of the matching records, data is erased in fields with the Field Clear setting turned on.

The Field Clear setting command is used to change the Field Clear setting flag between its two states, on and off. Initially, this setting is off for all fields of the form. When the command is selected, if the Field Clear setting is off, the message:

```
Field Clear setting is off. Turn on? [yes][no]
```

appears. A yes response can be used to turn it on and an no response can be used to leave it off. If the field Wipe out flag is on when the command is selected, the message:

```
Field Wipe out flag is on. Turn off? [yes][no]
```

appears. A yes response can be used to turn it off and an no response can be used to leave it on.

If data is entered into a field which has its Field Clear setting on, then the flag is automatically turned off.

Termination of editing can be done in one of two ways. The Exit command can ([Global] Function ⇒ Exit) be selected to avoid updating the records. If the information in the Insertion form has been modified, **texdave** prompts for confirmation:

```
Exit from Edit mode? [yes][no]]
```

If yes is chosen, the editing alterations are discarded and the records remain as they were. If no is chosen, Edit mode resumes.

If Write command is selected, the global substitution is performed, using the information in the form. Each of the records in the bin is updated automatically.

Numeric substitution ([Display] Edit ⇒ Numeric Substitution)

Perform a numeric substitution on the secondary key for all the records in the bin. This command is available only if the numeric secondary key matching option was initially selected and the secondary key is not a KE Texpress library item.

When this option is selected, the screen is cleared and a message appears:

```
Substitutions of the form - a * (X + b) + c
```

In the middle of the screen is the formula:

```
a * (X + b) + c
```

The user will be prompted with:

```
Enter a value for 'a':
```

The required value is entered, then the formula is re-displayed with the value entered replacing the symbol, 'a'. Similarly, values can then be entered for the symbols, 'b' and 'c', respectively. Again, on the entry of each value, the formula is re-displayed. After values have been entered, another prompt appears:

```
Do you wish to continue with the substitution? [y/n]
```

is displayed. If no is Chosen, the numeric substitution is avoided and the user is returned to Display mode. The records in the bin remain unaltered.

If yes is chosen, then for each record in the bin, the numeric value of the secondary key is substituted for the symbol, **X**, in the above formula, the formula is calculated and the result is copied into the field of the secondary key before the record is rewritten.

Delete current record ([Display] Edit ⇒ Delete record)

Delete the record from the data file. This operation removes a record from the data file entirely. When this option is selected, the message:

```
Delete record n? [yes] [no]
```

is displayed, where ***n*** is the relative number of this record within the current bin. Chooisng no can be used as an escape from the record deletion command, and returns the user to Display mode. If yes is chosen, the record is deleted and the user is returned to Display mode at the record following the one deleted.

Discard record ([Display] Edit ⇒ Discard record)

Discard this record from the current bin. The following message is displayed:

```
Discard this record from this bin? [yes][no]
```

If yes is typed, the record is removed from the current bin (but not from the data file). Display mode continues at the next record in the bin. Choosing no aborts the command.

View privilege level ([Display] Edit ⇒ View privilege level)

Display the record's privilege level. The message:

```
Record is at privilege level n
```

appears, where ***n*** is the record's privilege level. Typing any key returns the user to Display mode. This command is available only if the database has more than one privilege level.

Change privilege level ([Display] Edit ⇒ Change privilege level)

Adjust the privilege level of all records in the bin. The message:

```
Change privilege level of all records in bin? [yes] [no]
```

is displayed if there is more than one record in the bin. An no response can be used to abort the command. Otherwise, if yes is chosen, the message:

```
Enter new privilege level (between x and y):
```

is displayed, where ***x*** and ***y*** represent the lower and upper bounds for the new privilege level. A new privilege level can then be entered. Typing ↵ aborts the operation and induces no change. Otherwise the message a displays while the operation is performed.:

```
Updating privilege levels to n ...
```

At the completion of the operation, the message

```
New privilege level of all matching records is n
```

is displayed, where ***n*** is the new privilege level of all of the records in the bin. This command is available only if the database has more than one privilege level.

Copy Menu

Print/Copy all records ([Display] Copy ⇒ Copy all records)

Copy all records in the bin with any comments to a Unix file or send the information to the printer. The records appear as they do on the screen. Their comments are copied with them.

Print/Copy this record ([Display] Copy ⇒ Copy this record)

Copy the entire current record together with any commnets to a Unix file or send it to the printer. Press the TAB key to jump form one box to another, or from the Printer list to the File list. To select an existing file, move the cursor to the upper file list box, use the arrow keys to move to it, and press ↵. To create a new file, move the curor to the lower box and enter the new file name.

Interrupt

The interrupt command, generated by typing the **`Ctrl+C`** key (or whatever key is set up to generate an interrupt on your keyboard) can be used to interrupt any operation and immediately commence the next query.

When typed, this command causes the current operation to pause and the following message is displayed:

```
Commence next query? [yes] [no]
```

Choosing no indicates that the current operation should continue. Choosing yes indicates that the operation should be aborted and the next query immediately commenced.

There are occasions throughout **texdave** where the use of the interrupt facility is considered unsafe. During these times, the interrupt command is simply ignored.

Saving Changes ([Query] Function ⇒ Save)

The **texdave** program initially copies the entire data file to a temporary file, called dave*xxx* located in the user's tmppath, where *xxx* is the process id of the **texdave** command. All modifications to records are performed on the temporary copy and not on the original records. Thus to make the modifications to the original records (and thus make them permanent), it is necessary to save the data file. The following message is displayed:

```
Write changes to "davefile"? [yes] [no]
```

To seek confirmation before the command is performed. Choosing no aborts the command. If yes is chosen, the modified temporary copy of the records is written to the original file. While this occurs, the following message is displayed:

```
Rewriting "davefile" ...
```

At the completion of this command, the user is returned to Query mode.

If many modifications are made to the records during a single session, the query response time can degenerate. To improve query response time, regularly save the changes during the session.

Exit - texdave ([Query] Function ⇒ Exit)

To terminate a session with **texdave**, the Exit command is selected during Query mode. If no modifications have been made since the commencement of the session or since the last time the Save command was performed, then this command causes a silent exit from **texdave** and a return to the invoking shell.

If, however, modifications have been made to the records in the file, **texdave** responds with the message:

```
"datafile" modified. Really exit? [yes] [no]
```

where ***datafile*** is the name of the standard data format file specified by the user in the **texdave** command. A response of no aborts the exit command and continues with Query mode. A response of yes causes texdave to exit. Any modifications to records performed since the last time the Save command was performed (or the beginning of the session if the file has not been saved since) are discarded, and the original database data file is not updated.

Read Only Data Verification

If **texdave** is invoked on a dave file for which the user does not have write permission, then the program is said to be operating in **read only** mode. This mode can also be selected explicitly by the user by invoking **texdave** in the following way:

```
texdave -R dbname davefile
```

In read only mode, the dave file is not first copied to a temporary file and all commands which can change the data in the dave file are disabled. However, all other operations which do not affect the data work are available, as previously described in this chapter.

Index