

# PROGS

Professional & Graphical Software

Haachtstraat 92  
3020 Veltem  
Belgium  
Tel/Fax (016) 48 89 52

## DATAdesign The main program

Joachim & Nathan Van der Auwera  
PROGS  
PROfessional & Graphical Software  
©1990-1993

April 5, 1993

# Contents

<b>1 Introduction</b>	<b>3</b>
1.1 Motivation	3
1.2 History	4
1.3 Credit's due	5
1.4 DATA design engine	5
1.5 Loading the program	6
1.6 Application Programming Interface	6
1.7 Concepts	7
<b>2 General</b>	<b>9</b>
2.1 Principles	9
2.2 Method of working	9
2.3 Windows and keys	10
<b>3 Commands</b>	<b>13</b>
3.1 Next / Previous	13
3.2 Info	13
3.3 Again	13
3.4 Mark	14
3.5 Display	14
3.6 Status	14
3.7 Begin/first	15
3.8 End/last	15
3.9 Search..	15
3.10 Replace..	15
3.11 Filter file..	16
3.12 Sort file..	16
3.13 New field..	17
3.14 Erase field..	17
3.15 Duplicate record	17
3.16 Undo/Truncate	17
3.17 Delete record	18
3.18 Clear all mark	18
3.19 New record	18
3.20 Set MEMO field..	18
3.21 Clear field..	18
3.22 Stuff record..	18

3.23	Indexed find.. . . . .	18
3.24	View file.. . . . .	19
3.25	Quit program . . . . .	19
<b>4</b>	<b>Files</b>	<b>21</b>
4.1	Load file.. . . . .	21
4.1.1	Kind of medium-files . . . . .	21
4.1.2	Filenames . . . . .	21
4.2	Save file . . . . .	22
4.3	Save new name.. . . . .	22
4.4	Forget file . . . . .	22
4.5	Print records.. . . . .	22
4.6	Backup of file.. . . . .	23
4.7	Merge file.. . . . .	24
<b>5</b>	<b>Utility programs</b>	<b>25</b>
5.1	ConvExp . . . . .	25
5.2	OLDtoNEW . . . . .	25
5.3	ConvOLD . . . . .	26
5.4	ConvFIBk . . . . .	26
<b>6</b>	<b>Appendices</b>	<b>27</b>
6.1	Configuring DATAdesign . . . . .	27
6.2	Versions . . . . .	27
6.3	Statistics . . . . .	28

DATAdesign software and documentation are copyrighted with all rights reserved. No part of this documentation or the DATAdesign software may be copied, reproduced or stored on any electronic or other media except for personal use.

The *ptr\_gen*, *wman*, *hot\_rext*, *config* files are copyrighted by Qjump.

The *menu\_rext* file is copyrighted by Jochen Merz.

The *ConvOLD*, *ConvOLD.txt*, *Import*, *Import.txt* files are copyrighted by Wolfgang Lenerz.

In no circumstances will PROGS, PROfessional & Graphical Software, be liable for any direct, indirect or consequential damage or loss arising out of the use or inability to use this software and its documentation.

# Chapter 1

## Introduction

This is the manual for DATAdesign. We advice strongly that you read it as it contains some information which is very interesting to know, and because it may avoid many problems while using the program. Most enquiries we get about our programs could be solved by reading the manual.

This manual does assume that you are acquainted with the Pointer Environment. Don't worry if you aren't, we have included an introduction at the very end of this booklet.

### 1.1 Motivation

DATAdesign is actually a special kind of database because it is "free-form". This means that the lengths of the fields are not limited, and that fields can be added and deleted at any moment you want to. The main advantage for this is that you don't have to know in every detail how the data in your database will look like when you create it. For instance if you would create a database with four lines for the address and you come across someone who needs five, what can you do? This kind of problems will never occur in DATAdesign. On the other hand if you suddenly realise that you also have to keep the VAT number of some customers, you can just create a new field which contains just that.

Of course the fact that DATAdesign is a free-form database also has some more negative implications. It is the case that databases where all fields have a fixed length are faster to access. You always know where your record is, and if you change the record, it can never grow larger than it was, because all records are equal in length. On the other hand a fixed-length database can have shorter files, which also load faster, if all fields are almost filled, as they don't have to know how long a field is (it is always the same). But free-form databases can also be much smaller and more practical in some cases. Can you imagine what a database of letters would do in a fixed-length database, or even worse, a file which contains chapters of a book.

## 1.2 History

This is actually the third (and probably final) incarnation of DATAdesign. It is a program which has gone through a (r)evolution.

It all started with DATAdesign v1.00. This version of the program was not much more than a simple cardfile database. Of course it already used the Pointer Environment, but that says it all.

After almost a year, and mostly because Wolfgang Lencz asked for it, we split the original program in two parts: the main program and the engine. This had the main advantage that the DATAdesign database management system (dbms) was accessible by everyone, so that there finally was an alternative to Archive.

This however had two problems, we had to rewrite big parts of the main program, as we had to call all the routines in a different way, and because we also added some extra features like filter, mark, duplicate and shortcuts to access some routines. We didn't write it properly though, so DATAdesign v2 relied heavily on the internal structures of the engine.

On the other hand there was the engine (v1). We supplied it in two parts, the engine itself, and the SuperBASIC interface. The engine lacked some commands to access some of the embedded data, and the SuperBASIC interface was quite stupid. You had to explicitly type all the parameters, and parameters were quite type sensitive (don't pass an integer as a float).

About seven months later we released the second version of the SuperBASIC interface. It was a major improvement because the commands were more readable and because you could omit the parameters you didn't need. The interface was also more friendly about accepting parameters of different types, but it had problems with array elements. We also made sure that the files were accessed more easily, with the disadvantage of having more problems when accessing several files simultaneously in a program.

We now (about 16 months after that version) have release DATAdesign v3. It is now also sold in two parts, as not all DATAdesign users need the programming interface. We have now included the new SuperBASIC interface directly into the engine, and the engine is now v3, to make the sure the engine and main program at least have the same figure in front of the dot in the version number.

The DATAdesign main program is now finally a program which accesses the engine properly, without accessing the internal structures, and we have redesigned the window. Actually this is (another) complete rewrite of the program and we have tried to make it better then it ever was.

The major difference is actually contained in the engine. The SuperBASIC interface is now as it should. Very flexible with the parameters, and an improved method to handle errors. It is now also easier to access several files in the same program, thus, together with unique numbers for fields and records, creating a powerful relational database for the programmer.

More noticeable to the average user is the introduction of disk-based files and indexes. The first allows for large files to be accessed; this should prove very handy for all those people with large databases like doctors, lawyers and people who compile QLAW databases. The latter provides a FAST search in a program and better handling of sorted and/or filtered files.

The engine now also allows types to be given to a field, thus allowing numeric values and also raw data like screens or fonts or anything you want.

## 1.3 Credit's due

There are some people who have been a great help during the development of DATAdesign. To start we would like to thank our parents, as we wouldn't be programmers without them, they have always supported us and continue to do so. Thanks.

We would also like to thank Tony Tobby and his team at ~~Qump~~. They have made the task of programming a user interface much easier, and provided a powerful interface for the engine with the thing system. Jochen Metz on the other hand provided some practical windows for things like loading a file, reporting errors, confirmation requests,...

We would also like to thank some people who gave some help and good advice to improve the program and/or manual (no special order): Roberto Orlandi (Italy), Erik De Weert (Belgium), Ken Bain (England), Graham Evans (U.K.).

A special thanks has to go to Wolfgang Lenerz. Partly for beta testing this program, and mainly because he gave a lot of guidance in what the next step in the development of DATAdesign could/should be. He also wrote a utility program which is provided on the accompanying disk to convert v2 DATAdesign files to v3 and giving you the option to convert the data types. He also wrote a similar program for converting Archive export files.

Another special thanks has to go to Sohail Bhatti from QLAW. He uses the DATAdesign engine to compile its database and thus came across some problems that he reported and we tried to solve. He is a big fan of DATAdesign and tells this to others.

Yet another special thanks has to go to Dilwyn Jones from Dilwyn Jones Computing. As our U.K. dealer he saves us from a lot of work marketing our program and allowing us more time to concentrate on programming.

Thanks to you all, we need your help and support for this program (and any other program we (and others) write). So if you have any suggestions, don't hesitate to write or call us, and we can try to help you, and you can help all the other users of DATAdesign. Worse: if you find a "b-u-g", you have the moral duty to tell us when-where-and-how you found it, so we can help you all by correcting it.

## 1.4 DATAdesign engine

This is the part of the DATAdesign package that makes everything happen. Without it DATAdesign would not exist.

There are however some details which are important to everyone. After you have loaded the engine, it can't be used yet. You have to include the "ENGINE\_INIT" command for this. This is done because the engine creates a special job called "DATAdesign files" to store all the information contained in it. As you may want to load some more resident extensions after the engine, we could not initialise automatically.

The "ENGINE\_INIT" command is also important because it allows you to re-initialise the engine. This may be necessary if you accidentally removed the "DATAdesign files" job. However, you should NEVER delete the "DATAdesign files" job. It contains all the files which are accessed, so if you remove this job you will actually remove all files as well. This may cause problems for the jobs which were access-

ing these files, and therefore all jobs which were using a DATAdesign will also be removed<sup>1</sup>.

## 1.5 Loading the program

*Use a backup. We want to stress the fact that you must first make a backup from the master disk and USE THE BACKUP for loading.*

The easiest way to run the program is by inserting your backup of the DATAdesign master after a reset or power on, or you can load all the parts separately.

To use DATAdesign some resident extensions have to be loaded. This can be done with the following boot file<sup>2</sup>. Please note that the lengths of some of the files may be different.

```
100 base=RESPR(11684) : LBYTES flp1_hot_rext,base : CALL base
110 base=RESPR(14534) : LBYTES flp1_ptr_gen,base : CALL base
120 base=RESPR(10360) : LBYTES flp1_winan,base : CALL base
130 base=RESPR(22432) : LBYTES flp1_menu_rext,base : CALL base
140 base=RESPR(31408) : LBYTES flp1_engine_rext,base : CALL base
150 HOT_GO : REMark just to make sure you can still use alt-enter
160 ENGINE_INIT : REMark initialise the DATAdesign engine
170 EXEC flp1_DATAdesign
```

These lines may of course also be included in your own boot file. If all the resident extensions are loaded, then you can run DATAdesign with a line like.

```
EXEC flp1_DATAdesign
```

If you want you can also run DATAdesign with a file loaded (if you have toolkit 2), this can be done with a line like.

```
EXEC flp1_DATAdesign;'flp1_example'
```

If no device is specified, then the DATA\_USE device will be searched. You should not specify the ".ddf" extension.

If you have QPAC II then you can also define a hotkey and/or create a button with the next lines.

```
ERT HOT_RES('d','flp1_DATAdesign')
BT_EXEC 'DATAdesign'
```

## 1.6 Application Programming Interface

The DATAdesign engine can be programmed from SuperBASIC, C, Assembler. If you want to make programs using the engine you should buy the API (Application Programming Interface) from us. For more information, just write or call.

<sup>1</sup> Actually even the DATAdesign.engine thing will be removed, but it can be re-initialised with "ENGINE\_INIT"

<sup>2</sup> This boot file will not work if you have a JM QL. In that case use the boot file as supplied on disk.



## 1.7 Concepts

**file** A file is the entity which includes a set of related data to be used by the DATAdesign engine. When we use the word file in this manual, we don't mean a file as in "a file on disk", but we mean a file which is used by the DATAdesign engine. The conventional usage for file won't be used a lot in this manual, and the conventional usage will be called *medium-file*. A file may be on disk (*disk-based*), or it may be completely in memory (*memory-based*). When a medium-file is not used by any job in memory (so there are no buffers using the file (see later)), then this medium-file will not be called a file. Files are referenced by (hopefully unique) *filenames*. These filenames are case-dependant.

**buffer** A buffer is an entry-point to a file. It contains a copy of the *current record*. If the buffer is not a *read-only buffer* then the record will be *locked*, that is, unavailable to all other buffers using this file.

**record** Records are parts of a file which combine related data. If you go to a library and you want to find a book, you search the *register*, which is a database. In those libraries where you still have to find them manually, there will be a place where you can find a card for each book. Each of these cards is a record, and all the cards together are the file.

**field** Fields are subdivisions of records, and these subdivisions are available in all records. To use the example of the index at the library, the fields are the subdivisions of the cards, like *author, title, publisher,...*

**field type** In the DATAdesign engine, all fields are typed. Five basic types are provided and these types should allow you to put any kind of data you want in a field. It is up to the creator of the database to determine what a certain value in a field is supposed to mean.

type	char	usage
raw	none	graphics, fonts, ... can't be accessed in DATAdesign main program
char	none, \$	text
short	%	small integer values selections, statuses, ...
long	&	large integer values dates, ...
ieee	£	IEEE double (floating point) any numerical value

**initialisation** As you probably noticed it is not enough just to load the DATAdesign engine, as this only loads the engine and links the new SuperBASIC commands. The engine itself is not initialised as this creates a special job which contains all the actual data. This is for safety reasons, to make sure that no data can be released by accident. There are two minor hitches to this approach. First you have to explicitly initialise the engine. This can be done with the SuperBASIC command ENGINE\_INIT. Secondly, if you remove the *DATAdesign files* job, then the



engine and all the data will be removed. You should never do this. For memory protection reasons, when you delete the *DATAdesign files* job, all jobs which use the *DATAdesign* engine will also be removed.

**file-status** This is a special property which each file has. It indicates whether a file is disk-based or memory-based. By default all files are memory based when you create them. But that can be changed. There is no automatic switching between the two statuses.

**disk-based** This file-status is included for two reasons. Firstly to allow for very long files to be used, even files which are much longer than memory will permit. However there is always an index with references to the place which has to fit in memory<sup>3</sup>. Secondly it is the safest way you can work on a file. Even if a system-crash would occur for some reason, then a maximum of one record will be lost. There are a few commands which are actually not that safe, but it will be mentioned when they are discussed.

**inter-record-space** When a file is disk based and a change to a record makes that record grow a bit, then it would have to be moved to the end of the file as it would not fit in the medium-file at the old place. To prevent this from happening too often, you can make sure that there is always a bit of empty space after each record. This space is called the inter-record-space. This is not relevant when a file is memory-based.

Even when a large inter-record-space<sup>4</sup> is used, it may not be always be enough. That way large empty gaps will be created in the file. These gaps can be removed with *garbage collection*.

---

<sup>3</sup>This shouldn't be a problem, every record only takes 18 bytes in this index

<sup>4</sup>This is not advisable as it can waste a lot of disk space. We advice small values, e.g. 10

# Chapter 2

## General

### 2.1 Principles

DATA design is a standard Pointer Environment program, so everything can be done in the usual way. If you are not familiar with the Pointer Environment, there is a special manual at the end of this booklet.

In all sub-windows all items which activate another subwindow can be recognised because they end in "...".

In this manual keys which should be pressed are always shown like this <KEY>. When you should press two keys at the same time, it will be print like <KEY1-KEY2>. On the other hand, if we say <KEY1>-<KEY2> then you should first press <KEY1> and then <KEY2>.

### 2.2 Method of working

When you want to create a record in an empty database, you first have to define the fieldnames. This can be done by just typing the names of the fields in the edit-window, each on a different line. The type of the field will be defined by the first letter. If this is a special character, then the field will get a type accordingly, and the character will be removed from the fieldname. If the character was not recognised as denoting a fieldtype, then you will get a character field (and the first letter is not removed).

type	char	usage
char	none, \$	text
short	%	small integer values
long	&	large integer values
ieee	£	ieee double any numerical value

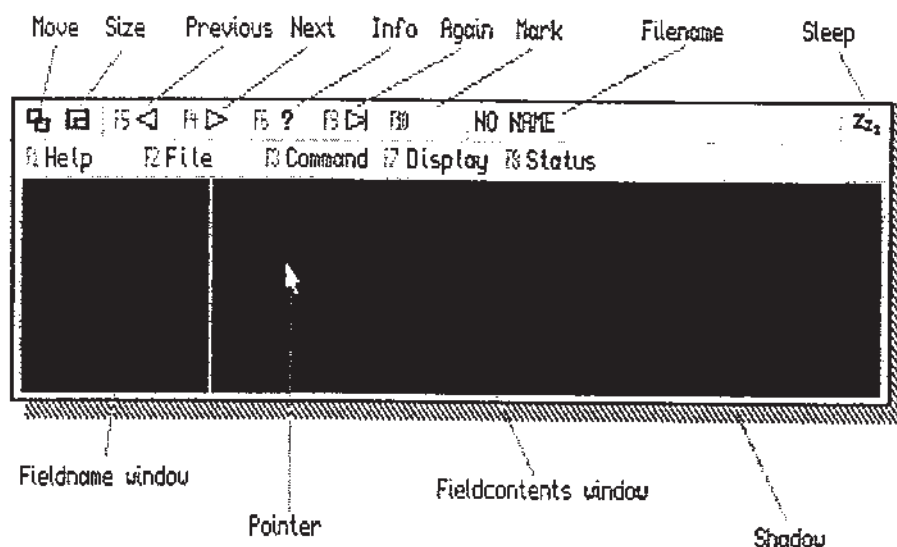
Fieldnames can be no longer than sixteen character (they will be truncated), and no two fields with the same name are accepted (only the first one will be created).

Don't worry if you have forgotten a field, you can always add (or delete) it.

If you then press <F4> or <F5>, the fieldnames will be moved to the fieldname window and you can start typing to create some records. The records themselves

can be put in the file by (again) pressing <F4> or <F5>. This will also give you the next (resp. previous) record in the file. When you start typing a new file, the next record will always be an empty one. However, when you are modifying a sorted file, the next record will be the record which is just after the one you just typed. In that case it may be more interesting to type <CONTROL-R> for 'New Record'.

## 2.3 Windows and keys



The DATAdesign window consist of three distinct parts which all have a different usage. At the top we have the part with all the command items. Some of those are standard Pointer Environment items, and some are specific for DATAdesign. Those will be explained further later in the manual. The only thing we would like to mention is that it is advisable to only put the program to sleep when there is no record in the edit window, as that record would then be locked. This is only important if there are other jobs in memory which access the same file.

The second part is the *fieldcontents window*, which is also called *edit window*. This part is actually used to type or edit all the data in the file. Typing in this window is quite similar to using a wordprocessor or editor. But here are some differences. If a line is wider than can be visualised, then part of the line will pan in the left margin while typing. When editing another line, the first part will be visible. This panning will only occur in the line you are editing. When typing you can use all printable characters and the following controlkeys.

<TABULATE>, <SHIFT-DOWN>	go to start of next field
<SHIFT-TABULATE>, <SHIFT-UP>	go to start of previous field
<ENTER>	new line in this field
<ESCAPE>	leave edit-window, get pointer back
<LEFT>	cursor left one character
<ALT-LEFT>	cursor to start of line
<CONTROL-LEFT>, <BACKSPACE>	delete left on character or add line to previous when on first character
<CONTROL-ALT-LEFT>, <CONTROL DOWN>	delete line
<RIGHT>	cursor right one character
<ALT-RIGHT>	cursor to end of line
<CONTROL-RIGHT>, <DELETE>	delete character under cursor
<CONTROL-ALT-RIGHT>	delete to end of line
<UP>	cursor up one line
<DOWN>	cursor down one line
<CONTROL-B>	Begin/first
<CONTROL-E>	End/last
<CONTROL-S>	Search..
<CONTROL-F>	Filter file..
<CONTROL-O>	Sort file..
<CONTROL-N>	New field..
<CONTROL-A>	Erase field..
<CONTROL-U>	Duplicate record
<CONTROL-T>	Undo/Truncate
<CONTROL-D>	Delete record
<CONTROL-C>	Clear all mark
<CONTROL-R>	New record
<CONTROL-M>	Set MEMO field..
<CONTROL-L>	Clear field..
<CONTROL-V>	View file..
<CONTROL-Z>	Stuff record..
<CONTROL-P>	Replace..
<CONTROL-X>	Indexed find..
<CONTROL-Q>	Quit program

Note that <CONTROL-C> is normally overwritten as key to switch jobs. You can get into edit-mode by a hit or do on the window, or by pressing <E>. Getting out of edit-mode can be done by going up on the first line or going down on the last line of the record.

The third part of the DATAdesign screen is the fieldname window. This window is used to show the names of the fields. It can also be used to delete a field or to change the name of a field. This can be done by a hit or a do on the fieldname you want to change or delete. You can then change the name. If you delete the name, then the field will be deleted. Please note that a name which is exactly sixteen characters long can't be edited this way.



## Chapter 3

# Commands

### 3.1 Next / Previous

These commands can be used to browse through the file, and this either forwards (next) or backwards (previous). They always display the next (resp. previous) record according to the order which is currently valid. It can also be used to

- include a new record in the file
- include the changes you made to a record in the file
- create the fields when creating a new file

Please note that "next" will display an empty record when you were on the last record. This empty record can then be used to fill in a new one. If you indicate "previous" when on the first record, then no other record will be displayed.

### 3.2 Info

Indicating this item will give you some information about the current file and current record. The window displays.

- FILENAME the name of the file you are working on
- FILEDEVICE the device where the medium-file is stored
- NO. OF RECORDS shows how many records there are in the file
- NO. OF FIELDS shows how many fields there are in the file
- RECORDS IN INDEX shows how many records are visible
- RECORDLENGTH show the length of the current record, no real importance
- RECORDDATE shows when the current record was last changed
- RECORDID shows the unique recordid for the current record

### 3.3 Again

This item can be used to repeat a search command.



### 3.4 Mark

All record in the file have a *mark status*. This item shows the mark status for the current record and also allows you to change that status.

A mark status is a status which is file-specific (so it is the same for all buffers which have access to that file). A mark status has a value between zero and 255. If the mark status is zero we say it is cleared.

The mark status of a record can be used to make selections which can't be defined in filter. You could in fact consider it as an extra field which can contain at most one (small) number.

### 3.5 Display

This command can be used to define which fields should be visible in the edit-window, and in which order.

All you have to do is indicate the fields you want viewed in the edit-window in the order you want them. Then press <ESCAPE>. Now only your selection of fields will be visible.

If you indicate 'all' in this window, then all the fields will be selected in the order in which they are displayd in the window.

### 3.6 Status

This command gives a small sub-window in which you can do three things.

You can change the file-status. This gives you the option to choose whether the data in your file is actually kept on disk or in memory. Keeping it in memory has a speed advantage, but keeping it on disk is safer in case of a power failure or system crash.

The other thing you can change is the inter-record-space. This defines a gap which is left between records on disk. This gap is used to overcome small variations in record size when you change something in a disk-based file. It is important as these small gaps may prevent some very large gaps from occuring in your medium-file.

There is also a *garbage collection* item in this window. This command should occasionally be used if you file is disk-based. It actually clears up all unwanted<sup>1</sup> whitespace in the medium-file. If the inter-record-space is too small, you should collect garbage more often. Please take care, you only collect garbage after making a backup of your file. If a power failure would occur during garbage collection, then your file may be lost.

**Whitespace** in a medium-file can be created because a record has grown a bit and doesn't fit in the file any more at the old place. The record will then be moved to the end of the file, thus leaving an empty gap. This empty gap will not be re-used unless the record which was just before it<sup>2</sup> grows a lot.

<sup>1</sup>That is whitespace  $\neq$  inter-record-space

<sup>2</sup>in the physical medium file, not in the file as accessed with Next or Previous

### 3.7. BEGIN/FIRST

## 3.7 Begin/first

This command can be used to go to the first record in the file. This will be the first visible record which is not locked by another job which accesses the same file.

## 3.8 End/last

This command can be used to go to the last record in the file. This will be the last visible record which is not locked by another job which accesses the same file.

Please note that this record will always be followed by an empty record which can be used to create a new one.

## 3.9 Search..

This window allows you to search for a string or numerical value in a particular or in all fields (with the given type). There is an item where you can type what value should be found.

If you want all fields to be searched, then all "suitable fields" should be visible in the scrollable window, and you should indicate the type of fields to be searched<sup>3</sup>. If you want to search in one field only, you just have to make sure the name of that field is visible in the scrollable field.

You can also indicate whether the file has to be searched from start to end, or from end to start (search backwards). When searching for a string, it is also possible to demand that the case of the letters is the same (match case).

When you activate this window with DO, then the value will be searched from the start (resp. end) of the file. You can then continue the search with 'again' (<F9>). If you press <ESCAPE> in the window, then you can immediately search starting from the next (resp. previous) record with 'again'.

Please note that the searching occurs without indexes because this is more flexible. On the other hand it is also slower.

## 3.10 Replace..

This window allows you to replace a string or numerical value in a particular or in all fields (with the given type). There is an item where you can type what value should be found, and what the new value should be.

If you want all fields to be searched, then all "suitable fields" should be visible in the scrollable window, and you should indicate the type of fields to be searched<sup>4</sup>. If you want to search in one field only, you just have to make sure the name of that field is visible in the scrollable field.

You can also indicate whether the file has to be searched from start to end, or from end to start (search backwards). When searching for a string, it is also possible to demand that the case of the letters is the same (match case).

<sup>3</sup>When you indicate 'integer', the value will be searched in both 'short' and 'long' fields, unless the value will not fit in a short.

<sup>4</sup>When you indicate 'integer', the value will be searched in both 'short' and 'long' fields, unless the value will not fit in a short.

When you activate this window with DO, then the value will be replaced from the start (resp. end) of the file. You can then continue the replace with 'again' (<F9>). If you press <ESCAPE> in the window, then you can immediately replace starting from the next (resp. previous) record with 'again'.

Please note that the searching in replace occurs without indexes because this is more flexible. On the other hand it is also slower.

Replace can only replace one occurrence of a string or numerical value per record. If you want more occurrences to be replaced, then you should start again from the start of the file.

### 3.11 Filter file..

This command allows you to select which record in the file should be visible, and which not. There are up to ten filter levels. You always edit the level which is indicated in the top scrollable window. Near the bottom there is another scrollable window which indicates the field this level works on, or that only previous levels are taken into account (if any).

Each level which is evaluated has a *levelstatus* (true or false). These statuses are combined by taking the first value, and then ORing (And not indicated) or ANDing (And indicated) the next level with it.

Using all the items in the window you can select whether

- a field is or is not cleared.
- a certain character (any of up to eight per level) is or is not present in the field (*place* = 0). Zero bytes are not evaluated.
- or is present at a certain place in the field (*place* > 0).
- a certain string (maximum eight letters long) is or is not present in the field (*place* = 0).
- or is present at a certain place in the field (*place* > 0).
- whether the mark status <, >, =, ≤, ≥, ≠ the given value.
- whether a value in a numerical field <, >, =, ≤, ≥, ≠ the given value (if *place* = 0 or element doesn't exist → *levelstatus* = *false*).

When the field is a text field, then you can also specify whether the comparing of characters should be done case dependent or not.

### 3.12 Sort file..

This window allows you to sort all the visible records in the file. If sort is switched off, records will be in the order of creation<sup>5</sup>. There are up to ten sort levels. Only when two records are the same (as far as sort is concerned) in all previous sort levels, will the next level be taken into consideration.

<sup>5</sup>this is actually not always true, but this is so unlikely that it is a safe generalisation, for more info, read the API.

When "Off / no sort" is indicated then sort is switched off (at least when you indicate DO).

You can also indicate the place. For numerical fields, this is the number of the item in the field which is sorted. If the item doesn't exist, then the record will be placed at the end of the file. When the field you want to sort is a text field, then this indicates the character which is used for sorting. When you have also indicated 'as text' then it is the number of group of eight characters. However when 'in line' is also indicated, then it will always be the first eight chars of the given line.

When 'as text' is indicated, then you can also sort case dependent with 'difference for case'.

You can also sort this level in 'reverse order'. The last item in this window is 'mark equal records'. When two records are still considered equal after considering all the sort levels, then they will get a mark value of 255 if this item was indicated.

### 3.13 New field..

When you want to add a new field to your file, you just have to call this command, and type the name of the new field. The type can again be indicated by the first letter of the fieldname (see General method of working). Fieldnames will be truncated to 16 characters, and an error will be reported if there already existed a field with the given name.

The fieldname has to be typed in a Read String window from the Menu Extensions.

### 3.14 Erase field..

After a while some fields in your database may become redundant, and you may want to delete them. That is exactly what this command does. There is however one problem. You can only erase a field in a file if this is the only job which has access to that file, as we can't delete a field in a record which is being edited by another job. If no fields can be removed, an error will be reported.

To use this command, just indicate the fields you want to delete.

### 3.15 Duplicate record

In some cases you need to add a new record to the file which is almost the same as another record which already exists. In these cases this command comes in handy. Just execute it on the record with the similarity, then you can make the changes you need. It will actually creates a new record with the same contents as the previously current. If you don't want this second record, you can always undo with truncate (see below).

### 3.16 Undo/Truncate

If you have made if faulty change to your record or to the mark status, then you can restore the original values of both with this command.

### 3.17 Delete record

To keep your database up to date you will occasionally (or often) have to delete a record, and this command allows you to delete the current record.

### 3.18 Clear all mark

This command will clear the mark status of all records to zero. This will have no effect on records which are edited by another job.

### 3.19 New record

Always creates a new record you can fill in. This command is functionally equal to the <CONTROL-E>-<F4> key sequence when in edit mode.

### 3.20 Set MEMO field..

All records always contain a "MEMO" field. This is an 'invisible' field, as it can never be displayed in the edit-window. It is always a text field which can contain some additional notes. This command allows you to edit these notes.

### 3.21 Clear field..

This command can be used to clear the contents a some fields. This will be done in all records which can be accessed, so not in the records which are locked because another job is editing it.

You just have to indicate the records you want to clear.

### 3.22 Stuff record..

This command can be used to stuff the current record. The record will be stuffed as displayd in the edit window, that is only visible fields and in the same order.

There are three ways to stuff the record. You can stuff it by line, block, or in the scrap. If you stuff by line, then all lines will be placed in the stuffer buffer in reverse order, so you can recall them (in the correct order) with a sequence of [ALT-SPACE] and [SHIFT-ALT-SPACE].

If you stuff as block then the entire record can be recalled by only one keypress (i.e. [ALT-SPACE]).

The last alternative is to store it in the Scrap, this is part of the Menu Extensions, and can be used in QD and some other programs.

### 3.23 Indexed find..

This command gives you an other way to search in a DATAdesign file. It has the advantage of being blindingly fast, always, no matter how long your file is.

Naturally, there are some disadvantages as well. This command can only be invoked if the file is sorted, and you can only search according to the first sort level. This means that it only searches a given place in a given field.

All you have to do is edit the search key. The search key should be of the same type as the field in the first sort level. When you press enter the requested record is displayed, or a 'not found' reported.

Please keep in mind that this command needs an exact match. If the field was sorted case dependant, then the search is case dependant. If the field was sorted as text, then the first eight characters of the given string must match. If the string you search for is not eight characters long, then the search key should also be shorter than eight characters.

Please also keep in mind that a field which is sorted by character is always case dependant.

### 3.24 View file..

This command allows you to get an overview of the file you are editing. All the visible fields are (partly) displayd in the window. The window can be scrolled and panned. Panning will change the fields which are displayd, and scrolling will change the records. Note that scrolling is a bit slow because the data has to be retricked from the file.

The window can be split (only horizontal) to show two different sets of fields. You can immediatly jump to a record with a DO on one of its fields.

### 3.25 Quit program

Can be used to leave the program. If you have changed anything since the last time you saved, there will be a confirmation request (Item Select).





## Chapter 4

# Files

### 4.1 Load file..

This command allows you to load a new file. This file can be selected with the File Select window in the Menu Extensions, or if you configured your program that way, with the Read String window.

#### 4.1.1 Kind of medium-files

DATAdesign always attempts to load or save three medium files. All these files should have the same name and be on the same device, the only difference is the extension (and the contents of course). The actual file (which has to be present) which contains all the records has the '\_ddf' extension, and is the file which the engine always saves, loads or backs up. Then there is also a file with a '\_ddm' extension which contains more information about the size of the window and especially which fields should be visible and which not. The programs also attempts to read an index file. This file contains all the records which are visible (filtered), and the order (sorted?). Other program which accessed the file should also update this index file or you may have some missing records.

When you think that the index may not be up to date, you can rebuild it with a "do" in Sort or Filter (whichever is valid).

#### 4.1.2 Filenames

The file name and device are normally stored separately in the DATAdesign engine. For this reason, we have to split the filename and device when loading. We do this by first removing any three letter extension that the name may include (if there is one), and then taking everything from the end till the last underscore ('\_') as the filename, and the rest at the device. This does however cause problems in two cases. Filenames which are exactly three characters long should always be typed with an extension (e.g. '\_ddf'). The other problem is that when the same file is accessed by another job which allows underscores in the filename will not be treated as the same file. This can cause problems with database integrity.

When there is another job which is accessing a file with the same as the one you are trying to load, then this job will access the same file, which may not be the

file you wanted. For this reason, it is vital that you, as a user, choose DIFFERENT NAMES FOR ALL FILES. Do note that filenames are compared case dependent, so 'address' and 'Address' are treated as two DIFFERENT files.

## 4.2 Save file

This command saves the current file, and the medium-file with the window information, and if the file is sorted or filtered, also a file with all the index information. The files will automatically be overwritten if they already existed.

## 4.3 Save new name..

Functionally the same as "Save file", except that you can now change the filename and device. Please also read the paragraph about filenames in "Load file".

If the medium-file already existed, a request is made whether you want to overwrite or not.

## 4.4 Forget file

This command allows you to remove the file you are working on so you can create a new one.

## 4.5 Print records..

This sub-window allows you to print your records. It is made in such a way that it should allow you to print labels and lists. For more complex jobs, a dedicated printing program should be written. The window will start printing when you indicate the PRINT item.

This command will always print all the fields which can at that moment be edited in the edit-window. So selecting which fields to print can be done with "Display".

**Current record** When this item is indicated, then only the current record will be printed. This item can only be indicated when there is a current item, and you haven't printed all (visible) records yet. If this you print and this item is not indicated, then all visible records will be printed.

**FF at end** Indicating this item makes sure that the page is ejected when the printing is finished.

**FF between records** If you want all printed records to come on a separate page, then you have to indicate this one.

**Fieldnames** You can instruct DATAdesign to include the fieldnames when printing. When they have to be printed, the left margin will be increased by 17 characters, and the fieldnames will be printed in the margin, followed by at least one space.

The fieldname will only be printed for the first field on a line.

**Device** This item allows you to change the device which has to be used when printing. It will always suggest the default –which can be configured– when the window pops up.

**Margin** You can indicate how many spaces should be printed at the start of each line by filling in a number.

**recordend** Allows you to add some newlines at the end of each record. This way you can be sure there is a gap of any size you want between two records.

**Maxlen** As fields can have any number of lines in them, you can limit the number of lines which have to be printed by filling in a value. If it shows no value or zero, then all lines will be printed. If there is a number then that number of lines will always be printed. Thus either limiting the lines, or filling with empty lines.

This item can be different for all fields. It is only valid for the field which is displayed in the scrollable window. But you can change the field in that window by scrolling it of course.

**Line separator** This command allows you to choose whether the lines in a field should be separated by a space, or printed on different lines (newline). When the lines are separated by a space, then at least the first line of the next field will also be printed on the same line.

This item can be different for all fields. It is only valid for the field which is displayed in the scrollable window. But you can change the field in that window by scrolling it of course.

## 4.6 Backup of file..

You can make a backup of your file with this command. Just edit the filename (you should actually only change the device), and a backup will be made.

When your file is memory based, this allows you to save the file to another device (or name) without the internal filedevice changing.

When your file is disk-based, this is the only way you can make a backup without releasing the file (i.e. making sure all jobs which use it stop doing so). It does however require you to keep the medium with the file in place, so you will have to backup to another device (or to the same disk).

Backup of a file only makes a copy of the root file, the index file and the file with the window details are not backed up with this command.

## 4.7 Merge file..

This command allows you to merge two files together. You just have to select the file to be merged.

The fields which have identical names (case dependant) will be put in the same place. Other fields will be added. If there are fields in the two files with identical names and different field types, one of the files may actually display garbage. The fields to be merged should not be longer than 8k.

## Chapter 5

# Utility programs

### 5.1 ConvExp

This utility program allows you to convert Archive export files to DATAdesign. All character fields are treated as such, and numerical fields are treated as IEEE doubles (floating point values). When you run the programs, all you have to do is types the names of the file to convert, and the name of the newly created file (including device if DATA\_USE device not suitable. The program can be run with a line like EXEC flp1\_Convert if the DATAdesign engine is loaded.

### 5.2 OLDtoNEW

OLDtoNEW is a program which allows you to convert your old DATAdesign files into files which are readable by DATAdesign v3. The operation is similar to Convert, you just have to type the names of the old and new DATAdesign files. The program will report it if the given file is new enough, or not a DATAdesign file.

Because all old DATAdesign files were text files only, all fields are automatically converted to text field. If this doesn't suit you, you should use ConvOLD (see below).

This program was originally written in C and needs the DATAdesign engine to be loaded. You can get the source of this program as it is included in the API.

**file size** When converting old DATAdesign files to new ones, you will notice that the size of the file will grow. There are two reasons for this increase in length. To start with DATAdesign v3 files have a long header in their files which defines the order in which text is compared in normal and reverse order, and a large area in which the fieldnames are defined. This header is always equal in length and has to be provided because files can be disk-based.

On the other hand each field which is not empty will also be a bit longer. This is because all fields need a fieldheader which states the length of the field and which field it is. The fieldid is included because not all fields have to be present in a record. This allows for adding of fields to the file. If this was not the case, then



the adding of a field in a disk-based file would mean that each record would have to grow. This would in most cases prove very difficult to do.

The `fieldlength` is also included because, contrary to old files, a field separator cannot be used as all values can occur within a field. This is especially true for the numerical and raw fieldtypes. Text fields don't suffer this problem, but are internally treated exactly the same as non-text fields.

### 5.3 ConvOLD

This is a program which does about the same as `OLDtoNEW`, except that it allows you to indicate the type of the new field. This program was written by Wolfgang Lenerz. A small user manual is supplied in the file `ConvOLD.txt`, which can be read in any editor (e.g. QD), or with a line like.

```
VIEW flp1_ConvOLD.txt
```

It is a compiled SuperBASIC program which uses the Pointer Environment.

### 5.4 ConvFIBk

This is a compiled SuperBASIC program which can convert any FlashBack file into a `DATAdesign` file. Because of the nature of FlashBack files, all fields are treated as text fields.

The source of this program is part of the API.

## Chapter 6

# Appendices

### 6.1 Configuring DATAdesign

As a true Pointer Environment program, DATAdesign can also be configured with the standard Config program from Qjump. There is however not much that can be changed. If you never used the Config program read the Pointer Environment manual in this booklet.

You can configure whether File Select should be used when loading files. This doesn't mean that DATAdesign can work without the Menu Extensions. It only means that you can choose always to type the filenames directly when loading.

The default printerdevice can also be changed to any device you want to, and if you want you can also choose which file should be loaded as help file (and on which device this file should be found).

### 6.2 Versions

- v3.00 Current version, can only work with v3.00 and higher. The program is finally not relying on the internal structures of the engine.
- v3.01 Small problem with filename after a sleep solved.
  - 'As string' added in Filter.
  - 'All' added in display.
- v3.02 Bug in edit window with [ctrl-alt-right] solved.
  - Replace finally added.
- v3.03 Some problems in replace solved.
  - Merge file added.
  - Stuff record added.
- v3.04 View added.
- v3.05 Small bug in fieldname window which sometimes made a fieldname invisible when editing it solved.
  - It is no long allowed to erase/clear all the fields.

- v3.06 Sort/Filter occasionally crashed the computer.  
 Filter will no longer display a record when none is visible.  
 Sort now automatically displays the first record.  
 Problem when converting QL float to doubles solved.  
 Edit window doesn't test mouse movement any more.  
 Edit routines adjusted for better bug-safety (hopefully).  
 IndexFind finally implemented (F.A.S.T.).
- v3.07 Some problems in View solved. You could scroll one too far, and the index bar was not updated when scrolling up (this also caused some other problems).

### 6.3 Statistics

These statistics are actually the statistics for the DATAdesign engine, as the main program actually does pose an additional restriction to the size of the visible part of a record (max 8k).

max no of records	no limit, except memory (18 bytes/record)
max no of fields	256
max no of files	no limit
max no of buffers	no limit
max no of indexes	no limit
max recordlength	only limited by memory
max fieldlength	only limited by memory
max no of sortlevels	10
max no of filterlevels	10

Introduction to  
Pointer Environment  
&  
Menu Extensions

Joachim & Nathan Van der Auwera  
PROGS  
PROfessional & Graphical Software  
©1993

February 21, 1993

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Motivation . . . . .	3
1.2	Loading . . . . .	3
1.3	Concepts . . . . .	4
<b>2</b>	<b>Pointer Environment</b>	<b>7</b>
2.1	Pointer Interface . . . . .	7
2.2	Window MANager . . . . .	7
2.2.1	General . . . . .	7
2.2.2	Standard keys & items . . . . .	8
2.3	Hotkey System II . . . . .	9
2.4	Config . . . . .	9
<b>3</b>	<b>Menu Extensions</b>	<b>11</b>
3.1	File Select . . . . .	11
3.2	Item Select . . . . .	12
3.3	Read String . . . . .	12
3.4	List select . . . . .	12
3.5	Report Error . . . . .	12
3.6	View file . . . . .	13
3.7	Button . . . . .	13

This documentation is copyrighted with all rights reserved. No part of this documentation may be copied, reproduced or stored on any media except for personal use.

The *ptr\_gen*, *wman*, *hot\_rext*, *config* files are copyrighted by Qjump.  
The *menu\_rext* file is copyrighted by Jochen Merz.

Though much care was taken in the production of this manual, in no circumstances will PROGS, PROfessional & Graphical Software, be liable for any direct, indirect or consequential damage or loss arising out of the use or inability to use this software and its documentation.

# Chapter 1

## Introduction

### 1.1 Motivation

We have written this manual as a separate part because there are people who have already used programs which run under the Pointer Environment and Menu Extensions. These people don't have to read this manual as they will probably know it already.

The Pointer Environment and Menu Extensions are the way ahead for the QL. Most of the better new software for the QL uses it, or at least is 100% compatible with it. The Pointer Environment has the advantage of offering true multitasking and gives you the possibility to write pointer driven software, so you can use a mouse if you want to, with easy menus which can be accessed by a key or with the pointer. Using the Pointer Environment in a program interface has the advantage of being user-friendly. You can easily get used to it, and once you know the Pointer Environment, you know how to operate any program which uses it even before you open the package.

The Menu Extensions are actually just an extension to the Pointer Environment, to give standard windows to some common menus. The most important one is definitely File Select, which makes sure that you can select a file to load or merge with a directory, so that you don't have to remember any filenames.

### 1.2 Loading

The Pointer Environment and Menu Extensions are a set of resident extensions. This means that these routines should be loaded at start-up (power on or after a reset), and that they should remain in memory until the power is switched off or until the next reset. This also means that these extensions should be loaded into RESPR area. You can load the Pointer Environment and Menu Extensions by including the next lines somewhere at the start of your boot file.

```
base=RESPR(14534) : LBYTES flp1_ptr_gen,base : CALL base
base=RESPR(10360) : LBYTES flp1_wman,base : CALL base
base=RESPR(11684) : LBYTES flp1_hot_rext,base : CALL base
base=RESPR(22432) : LBYTES flp1_menu_rext,base : CALL base
HOT_GO
```

Please note that the lengths of the files may be different, this depends on which version you use. If you have Toolkit II you can also use these lines.

```
LRESPR flpl_ptr_gen      : REMark load the Pointer Interface
LRESPR flpl_wman        : REMark load the Window MANager
LRESPR flpl_hot_rext    : REMark load Hotkey System II
LRESPR flpl_menu_rext   : REMark load Menu Extensions
HOT_GO
```

This is actually easier as you don't have to worry about file lengths.

The `HOT_GO` command should be included to make sure that you don't lose the last line restore provided by toolkit 2 (alt-enter).

Please note that some of the files have to be loaded in a distinct order. You have to load `wman` after `ptr_gen`, and you have to load `menu_rext` after `hot_rext`. It is also possible that some other resident extensions have to be loaded before all these, in particular extension which change the screen driver like `Lightning` or `SpeedScreen`.

### 1.3 Concepts

**window** A window is a part of the screen. It can usually be recognised because there is a border around it. A job (=a running program) can have many windows in it, but all these windows must lie inside the *outline*. In most Pointer Environment programs the outline can be recognised by the shadow around it.

**pointer** Most Pointer Environment programs have a pointer. The pointer is the thing which moves when you move your mouse. If you don't have a mouse, it usually is the thing which moves when you press the cursor keys, but not always. The pointer may have the same shape and behaviour as the cursor, but a cursor can't be moved with a mouse. the pointer has the general advantage that it can have any desired shape and size. It is also possible to change the shape over time, thus creating something that looks like a cursor, or a walking person or anything you want.

Most application programs don't allow the user to set the shape of the pointer, but they often change it themselves to show you what kind of operations the pointer can be used for at a given moment.

**item** An item is a part of the window. It usually contains some text or a small drawing. An item can be recognised because a border appears around it when the pointer is on it. This border is removed when the pointer is moved to another part of the screen.

Items can have three distinct statuses.

**available** An item is available when it can be indicated by a hit or do, but isn't yet. Such items can be recognised because they usually fit into the general look of the window.

**selected** An item can also be selected, which means that the accompanying action will be taken later or is going on. Selected items can be recognised because they are highlighted in some way.

**unavailable** The last possible status is that an item can't be selected. This can be because a certain action can't be executed when the program is in a certain state of operation, or because the action isn't included in the program yet or similar. Unavailable items can be recognised because they usually aren't 100% clear.

**hit** Two of the most important types of input to a Pointer Environment program are a hit and a do. They are both ways to indicate items in a window. A hit is caused by pressing <SPACE> or the left mousekey. A hit changes the status of an item from available to selected or vice versa. Some items are immediately invoked when hit (e.g. a sub-menu), but this depends on the nature of the item.

**do** A do is quite similar to a hit, except that it always changes the status of the indicated item to selected. Usually<sup>1</sup> a do also invokes the item. This usually makes a do equivalent to a hit on the item and a hit on a "DO" item in the window<sup>2</sup>.

You can also invoke a do when you press <ENTER> or the right mousekey when the pointer is not on any item.

**underline** Most items in a Pointer Environment program can also be selected by another keypress than a hit or do, and this has the advantage that the pointer doesn't have to be on the item. These keys can usually be recognised because they are visualised with a little line under the letter in the command. For some items this is impossible. Those items often have an indication of the key which has to be pressed just in front or above them. This is mainly the case for items which can be called by pressing the function keys. This method of selection is equal to a hit.

---

<sup>1</sup>Not all items support this, it depends on the programmer(s), in fact some items even treat a "do" exactly the same as a hit

<sup>2</sup>In some cases there is no "DO" item, but only an "ESC" or "OK" item. In these cases a do usually invokes the "ESC" resp. "OK" item.



## Chapter 2

# Pointer Environment

### 2.1 Pointer Interface

This is the part of the Pointer Environment that makes sure that multitasking works as it should, making sure that a window is completely visible when printing or drawing in it. This part also makes sure that the pointer actually exists.

The pointer interface is contained in the *ptr\_gen* file.

It is the Pointer Interface that controls the proper handling of jobs (programs) and their windows. There are several ways to switch jobs. The first —and traditional— method is by pressing <CONTROL-C>, and thus running through all available jobs. The other method —which only works if the program which is currently using the screen doesn't cover the entire area— is by moving the pointer to a program which is partly visible. This program can then be selected by a hit or do. If you select with a do then a wake is also executed (if it exists, see later). If the program you want to switch to is not partly visible, then you will have to use <CONTROL-C> or a special program (e.g. Pick in QPAC II).

### 2.2 Window MANager

The Window Manager is the most visible part of the Pointer Environment. This is what allows programmers to make their programs to look and feel like any other Window Manager program. This common look and feel is even stronger because most Pointer Environment programs also use the same colours for the various parts of the menus and the Menu Extensions.

#### 2.2.1 General

Most menus are built from items and information objects. Information objects are there just to give the user some extra guidelines on how to use the program or some extra information like the position on a page, the name of the file which is edited or similar.


There are also (a lot of) items. Some stand on their own (loose items) and some may be grouped (in an application window). When items are grouped, it is possible that there are too many items to fit in the part of the window which is reserved

for them. In that case the application window will become scrollable or pannable (or both). There will appear some arrows at the border of the application window. When you hit one of these arrow items, the window will be panned or scrolled by one item. If you "do" on such an item, the window will be panned or scrolled by as many items as fit in the window (minus one).


It is also possible that there is a pan or scroll bar apart from these arrows. Such a bar tries to indicate what part of all the items is visible in the application window. By hitting somewhere in the bar, you can indicate which part should be made visible.

### 2.2.2 Standard keys & items


One of the main reasons why programs look and feel the same is because so much is done in the same way. For this, many standard operations have a common item.

 move This item is use to move the entire window to another place on the screen. The pointer will then get the same shape as the move item. You can now move the pointer and press <SPACE>, <ENTER> or any of the mousekeys. The window will then get the same relative movement as you gave the pointer. This command can be cancelled by pressing <ESC>.

Move can also be executed by pressing <CONTROL-F4>.


 size This item can be used to change the size of the window. Depending on the program this item can be handled in two ways. The window can be resized immediatly. This usually happens when the program only has a couple of distinct size it can handle. In the other case the window usually has a variable size. Then the pointer will get the shape of the item and the relative movement of the pointer indicates the change of size for the window. Please note that the bottom-right corner of the window remains in place. So moving the pointer to the left or top will increase the size of the window. Moving the pointer to the right or to the bottom will decrease the size of the window.

Size can also be executed by pressing <CONTROL-F3>.

 sleep This item can be use to make your job sleep. This means that your program will change appearance, becoming much smaller. This has two advantages: the job will use less memory, and there will be more space on your desktop (screen). The new appearance is called a *button*. It is a small menu which only contains the program name. If you hit on the item then nothing will happen (or if you don't have QPAC II, you will probably be able to move the button).

When you do on the item, then the program should return to the same status as before you put the program to sleep.

Sleep can also be executed by pressing <CONTROL-F1>.

 wake This item, which is not available in many programs, give you the option to of giving the program the option to refresh the screen. For instance, the directory will be re-read when you are in a window which shows a directory.

Wake can also be executed by pressing <CONTROL-F2>.

## 2.3 Hotkey System II

This part of the Pointer Environment implements the THING system. It is not important to know much about this unless you are an expert user. All you have to know is that it allows you to load general extension routines so that they can be accessed by any program. This is used for (amongst others) the Menu Extensions, the DATAdesign engine, some device drivers in the SMS2 system, ...

Hotkey System II also replaces the <ALT-ENTER> and altkey commands in toolkit 2 and adds some new ones. We will not give more details about all these options, as they are mostly used by people who already know the Pointer Environment quite well.

## 2.4 Config

This is a utility program, which is often used in combination with Pointer Environment programs to allow you to set some defaults in the program. This program can be called with a line like

```
EXEC flpl_Config
```

The program is quite straightforward to use. It will first prompt for the name of the program you wish to configure. It will then ask you what you want to configure and the new values. These new values can usually be set by typing the new value, or by cycling through all the values with any key and confirming with <ENTER>. The program will clearly tell you what to do and you just have to follow the given guidelines.

## Chapter 3

# Menu Extensions

This part of this documentation is mainly written by Jochen Merz as a user guide to the Menu Extensions.

### 3.1 File Select

The FILE SELECT window is always shown when the user is required to enter or select a filename. Here you can enter the filename either directly or edit a suggested one by selecting the menu option directly beneath the request.

Beneath this are two menu options with which you can recall the contents of the HOTKEY buffer and all previous contents. Just select the menu option and the contents will be written to the "suggested" area. Confirm with OK and the input will be accepted by the system. You can also edit the name, of course. The rest of the window concerns the current drive. Depending on the size of the window, one or two sub-windows are shown. If you only see one sub-window, this will contain the filenames and sub-directories. If you see two windows, the right hand (larger) one will only show the filenames, the smaller one on the left only the subdirectories. In these windows all the files are sorted alphabetically. The files will be taken from the current drive and must all have the correct ending (if any). The endings for sub-directories are ignored, because subdirectories don't really have endings. Now you can edit the drive and/or the ending.

If you press <ENTER> at the directory menu option, a further window is overlaid, from which you can select pre-defined devices and sub-directories. If you just select a directory, the list of files will be updated. You can also "update" it with a wake.

If you press <ENTER> at the endings menu option, this will be deleted if it wasn't already empty. This is easier and faster than having to select it and then deleting the four characters. If it was already empty, then a window overlay will show some suggested endings.

Above the current directory is a list of available devices, e.g. MDV or FLP. There are also drive numbers from 1 to 4 listed. To select RAM1., press <R> and 1, and the directory window displays RAM1..

Behind the directory name you will see an arrow "←". By selecting this option you can retrace a step back along the subdirectory tree without having to edit

anything. So if you're in directory `fp1_paul.texts` and select this once, then you get to `fp1_paul_`. The next time you select it, you get `fp1_`. If the current drive has true subdirectories (e.g. Miracle's Harddisk or the QL Emulator drivers) then you'll find the subdirectories of the file names marked with a "`→`". As already mentioned, subdirectories are always listed, the endings don't have to correspond. If you select such a subdirectory, then you'll "get in it", i.e. the name will be taken over for the directory and the file list read in again.

But to get back to the list of files: you can select any file you like. `<SPACE>` accepts the name as "suggested". `<ENTER>` takes the name and carries out an OK automatically. If the window is too small to show all the suitable files, the normal scroll arrows will appear in order to scroll the next batch of names up the screen. You can also select files or directories by pressing the character which is in front of the name.

At the right you will see a scrolling bar. Move to this area, press `<SPACE>` and the area will be shown relative to the total area. Press `<ENTER>` and the window will split, enabling you to control the two parts independently from each other. Move to the split and press `<ENTER>` to join the window together again.

You can also pre-select the eight different subdirectories suggestions in the Directory Select menu. Make the necessary changes to the `menu.txt` file with Config.

If the program which calls the File Select extension has a window which is not big enough to show the File Select window, then the filename will be prompted with the Read String extension (see below).

## 3.2 Item Select

When you see a window with one to three menu option, you can make your selection by pressing `<SPACE>` or `<ENTER>`. You can also press the first letter of the option.

## 3.3 Read String

You are asked to enter a string or filename. Under certain circumstances you may be offered a suggested name. You can either press `<ENTER>` to accept the suggestion, edit it as usual using the cursor keys or just enter a new string.

## 3.4 List select

This window is in fact very similar with Item Select. The only differences are that there can be more than three items in the window, and that the window can be scrollable if not all items fit in the window.

## 3.5 Report Error

The only thing you can do here is indicate the "OK" item to show that you've taken notice of the reported error.

### 3.6 View file

This window enables you to view a file. You can scroll one line with a "hit" in the view-window. You can scroll a page with a "do". Waking the window lets you start again by viewing the file from the beginning. Selecting WRAP causes any line, which exceeds the permitted width, to be continued on the next line, preceded by a →.

### 3.7 Button

If a program is in Button mode, then you can wake this up by moving to the button area and pressing <ENTER>. If the button is not positioned inside the button frame, you can move the button by using <SPACE>.