A proper explanation when you need one

Harold Thimbleby
Middlesex University
Bounds Green Road
UK-London N11 2NQ
harold@mdx.ac.uk

Peter B. Ladkin
Technische Fakultät
Universität Bielefeld
D-33501 Bielefeld
ladkin@techfak.uni-bielefeld.de

October 1, 1998

Abstract

Quality program design has received considerable attention from the software engineering community. Quality user manual design has received considerable attention from the human computer interaction community. Yet manuals and systems are often independently conceived, and thus do not well complement each other. This paper shows one method of easily obtaining correct and complete user manuals guaranteed to correspond with the system they document. The method has considerable merit for improving interactive systems design.

Keywords: User manuals. Interactive systems programming. Iterative design. Safety.

1 Introduction

We illustrate our approach by means of an example. We take a contemporary interactive system, a domestic fax/answerphone machine, which we have reverse engineered to a functional specification; we show how to derive a quality user manual from this 'raw' specification. The primary property of our approach is that the manual is guaranteed to be complete. (Inevitably, there are certain limitations to the guarantee, which we will discuss below.)

In cases in which the interactive system itself may also be derived from the specification, the correspondence between the interactive system and its manual is generally ensured.

The soundness of user manuals is itself important. However an important contribution of our work is the great efficiency the method permits for iterative interface design (cf. [6]). Because the manual is generated rapidly from the interface specification, users understand the system much earlier. They can therefore provide informed and useful feedback, which may result in a modified design, which in turn results in a modified system and an automatically updated manual. The turn-around time is thereby significantly reduced, yet the quality improves. When manufacturers are anyway trying to shorten product design times, the method of user manual generation we advocate could have a major impact on improving interactive systems quality.

Specifically, we automatically create a *skeleton manual* consisting of mechanically generated explanations, which, with our presentation system, a technical author transforms (in a flexible but regulated way) into a natural language manual. At any point the manual is guaranteed to be complete, even though it might consist partly of finished sections and partly of skeleton information. If draft manuals are printed, they contain quality control and auditing information, such as the times when particular sections have been checked out, and automatic summaries of actions on the technical authors (e.g., that certain sections need revision). A complete manual can be created at any stage from this information, as a combination of skeleton and fleshed-out writing. Our prototype delivers manuals as printed documents, but they can equally well be generated as hypertext or on-line manuals if required.

Correctness of the fully- or partially-completed user manual depends on: (*i*) whether the specification is correct; (*ii*) whether the algorithm generating the skeleton information is correct; and (*iii*) whether the technical writer faithfully reproduces the meaning of the 'bones' of the skeleton information. Compared with traditional manual generation techniques this reduces the ways in which errors can occur. Moreover, the technique can audit modifications of the system or manual texts, and hence form part of a larger quality process.

Our method breaks down the process of manual generation into logically separate but practically cohesive modules. Other advantages that accrue are:

1. *Efficient translation.* Multiple manuals may be implemented—for example in French, English and German—from a common skeleton;

2. *Flexibility of organisation.* Different sorts of manuals may be implemented describing the same device, based on different methods of generating the manual skeleton structure. (The manuals might also be automatically cross-referenced for convenience.)

3. *Version control.* If the specification of the system is changed, those parts of the manual affected (and only those) are automatically indicated for revision.

4. *System and manual evaluation.* The skeleton manual can be quantifiably evaluated, for example, on its length or other measures of complexity [1, 13, 14]. Given the ensured correspondence between system and manual, evaluating the manual relates closely to evaluating the usability of the system.

5. *System animation.* Prototype systems can be animated (demonstrated in action) by conventional methods from their specification. Animations may be supplemented with running natural language commentaries.

6. *Iterative design.* Both users and technical authors may quickly understand the detailed design of a system directly from its specification via the manual. They can therefore contribute to the design at this point, with the usual advantages of saved time, greater comprehension, and resulting improved usability of the system.

7. *Improved adequacy-checking.* Arbitrariness in a design is much easier to spot (and hence to correct) when it is explained [3, 10]. When explained automatically and immediately from a specification, needed corrections are identified close to source and thus the correction cycle may take less time and effort.

8. *Interactive help.* There is no reason why user help must be restricted to paper. Manuals can be interactive, context sensitive, adaptive—with guarantees that the information is correct even after system modifications [13].

9. *Illustrations.* Schematics and other helpful diagrams can be constructed, just as here we construct skeleton text [13]. Similar assurances of correctness could be available.

10. *Safety information.* Skeleton manuals need not be restricted to 'how to' information, but can include any information derivable from the specification. Of particular interest is safety information—because it must be accurate for legal and other reasons. The fax machine has no safety-critical features of which we are aware. However, it does have undocumented features, which would be unacceptable for safety-critical systems. Our method avoids this possibility rigorously, which an unautomated method cannot likewise guarantee.

11. *Separation of tasks.* Although we concentrate in this paper on the coordination of the manual with the system specification, the logical separation of these tasks leads to considerable flexibility. We use a markup language (LaTeX; HTML has also been used successfully) so typographic details and presentation of the manual are separated from its content; markup also permits separation of the manual itself from the version control information generated automatically; it allows the manual to be combined and arranged according to different criteria; writing the English (or French, ...) version of the manual is separated from deciding the meaning, which is generated automatically; defining the specific sections and their order in the manual is separated from defining its content (see below); and so on. Thus decisions about the manual design can be made by the appropriate domain experts (e.g., typographers, human factors experts, native speakers) independently, yet with assurance that correctness is maintained (subject to the provisos above).

12. *Low Cost.* Apart from the actual manual composition and writing (which must be done anyway), and the system specification (which must be available anyway) the overhead of our method is negligible—in fact, about a dozen lines of program (which could be re-used for further manual developments).

In the present study, we generate a function-oriented manual, although this may not be the only sort of manual required by users. A function-oriented manual is generally required for reference. One cannot hope to obtain a good manual of *any form* unless there is some guaranteed way to generate some correct and complete function-oriented manual—a minimal manual may be a worthy goal, but it must be based upon a correct function-oriented manual.

2 Background

Explanation and interaction have, historically, been separated. Devising good manuals and interactive help is difficult. Devising good interactive systems is difficult. Generally, writing manuals is left to technical authors, and implementing programs is left to programmers. Sometimes HCI specialists improve the manuals or the systems; rarely is the relation between the manual and system directly addressed. This is strange, for—one might assume—the correctness and completeness of a manual with respect to the system it documents is necessary for a user to be helped. (Manuals can legitimately contain constructive fictions to improve explanation; however, the technical author must distinguish deliberate fiction from accidental fiction.)

Here are some typical consequences of this separation:

- The Sagem fax/answerphone, currently marketed by BT as the DF200, has an incorrect manual. (We use the DF200 as the example below.)
- The Apple Newton has a feature promoted as an intelligent assistant, which is in fact little more than an on-line hypertext user manual. The programming language of the Newton, NewtonScript, provides no features for constructing the text or organisation of the assistant, which has to be done separately.
- Apple's Macintosh operating system version 7.5 has a broadly similar help feature [7]. The Macintosh permits users to modify the system (e.g., to move files); the help system cannot cope with such changes. Ironically this is an occasion where help is especially desirable: when a user has a non-standard system configuration!
- An analysis of a fragment of one version of the Flight Crew Operating Manual of the Airbus A320 showed that it contains ambiguities, and that it is an incomplete description [5]. The operator's manual is a legally required document according to which pilots fly the airplane, and is in this sense safety-critical. (Note that we are not claiming that the manual's deficiencies are safety critical—but what are the guarantees?)

In some areas considerable progress has been made. In expert systems it is normal for the explanation to be generated as a result of performing a query: the technique exploits the close connection between rules and their natural language interpretations, which are composed into substantial explanations as queries are processed. 'Literate using' has been suggested as a reliable method of generating examples for user manuals [9].

2.1 Programming paradigms

We have elsewhere [11] defined a 'programming paradigm for P' as a programming notation that supports P but that does not mention features of P. For example, Prolog programs do not mention resolution theorem proving in order to do it; thus Prolog is representative of the programming paradigm of logic programming. On the other hand: to get a Prolog program to do something in a particular sequence is quite tricky, whereas in an imperative language such as C or Pascal, sequencing is part of the programming paradigm and is trivial.

Thus, the paradigm—what a programming language provides but does not talk about—cannot be subverted by the programmer except by a deliberate effort. "What we cannot speak about we must pass over in silence." [15] Conversely, what is *not* in the paradigm is easily got wrong.

Our method can be seen as a forerunner of what might be called the manual-oriented programming paradigm, where the program for an interactive system does not mention the user manual, yet one is there nonetheless. Because nothing mentions the manual (at least up to the polishing of the skeleton text), the paradigm ensures the completeness and correctness of the manual. (Given the reputed aversion of programmers to writing documentation this is highly desirable!)

The closest example to manual-oriented programming is literate programming, but this does not support any given relation between the executing program and its user documentation, as our method does.

The Apple Guide system mentioned above (*op. cit.*) is an example of an ordinary system attempting to emulate manual-oriented programming. It does so with difficulty and with no guarantees. Interactive systems are written in Pascal and send signals to the Guide; the guide examines an independent database and presents text to the user. There is no paradigmatic correspondence between the material in the database and the Pascal program signals; moreover, since a Pascal program *constructs* user interfaces rather than defining them, there is no particular relation between the signals and the features of the user interface. Although a good user manual may be produced by these tools, it is a programmer's nightmare. One consequence is that if a user or technical author has an insight into improving the structure of the interactive system, the programmer would be reluctant to jeopardise the considerable investment to date in getting the manual to work at all.

With manual-oriented programming, the effort required to produce a quality manual is significantly reduced, and the management problems associated with manual correctness are eliminated. The recommended HCI procedure of iterative design becomes reliable rather than a burden to be avoided.

3 Method

The first author reverse-engineered a specification of the DF200 fax/answerphone [2]. (In commercial work, the unreliability of reverse engineering can be avoided since, in principle, the system specification is directly available.)

We then *automatically* generated from the system specification simple English explanations of how to achieve every function the DF200 supported. (For simplicity, we arbitrarily excluded communications facilities such as normal telephone operations.) This constituted the skeleton manual. We might also have automatically generated other information, such as: commands that are easily invoked after others; what preconditions are necessary to achieve before performing commands; safety information related to the various sections of the skeleton manual. In this study such refinements were not necessary.

The skeleton English descriptions are then displayed by a presentation system, an interactive aid which presents the manual writer with two main window fields per function supported by the device: the initial English text that has been generated, and a separate field for the manual writer to write the same or equivalent facts in refined English or other language.

The aid keeps track of when a field is edited, and supports a notion of 'checked' manual entries, which are then dated by the system. The aid provides a number of convenient interactive facilities (e.g., searching, cut and paste) that need not concern us here.

The aid then generates a specification of a polished English manual. At this stage, if technical authors had contributed nothing, the 'polished' manual would merely contain the skeleton text plus annotations that every section was unfinished. Typically, however, technical authors would flesh out the skeleton. In this case, the aid keeps track of what must still be worked on, and also permits the skeleton text to change subsequently (thus generating further actions for technical writers).

If the system specification changed after a manual was completed, the automatically generated revised manual would contain both the original English text alongside new skeleton text. This is intended to guide the technical authors to revise their own manual entries, or perhaps even to negotiate again with the system specifiers.

A LaTeX version of the manual is generated from the manual specification. The LaTeX manual includes information about checking-out times and so forth, and includes a summary of unchecked entries and other potential anomalies. As usual, LaTeX can generate tables of contents, indexes and other convenient features—again, which are guaranteed to be correct.

Since there are many different methods of organising a manual, we provide for the topological sorting of manual sections. The manual writer may group functions together under appropriate headings and require that certain function groups are described before or after other groups as appropriate. This is done conveniently in the manual writing aid. Separate sections may also be combined.

Finally, LaTeX macros are used extensively and included for all types of text generated so that the presentation of the manual is under complete typographical control. These macros can be edited if desired; for example, a draft manual could include margin notes or footnotes, as preferred, showing dates of section checking, which a production manual would not include.

If the system specification is modified, it can be re-read by the interactive part of the manual writing system. Any sections that have changed their skeleton text are unchecked. These sections are easily found automatically in the interactive aid, and are flagged clearly if they are used in a LaTeX document.

3.1 Implementation

We have implemented a rapid prototype of the system. The skeleton generation and manual generation parts are a short Prolog program. The user interface for the manual writer was implemented in HyperCard. The total implementation effort for the presentation system took about three days, including all discussions and modifications.

Prolog was chosen from convenience and reliability: we want guarantees that the manual is correct, and what better way to do that than in a programming paradigm where specifications and executable programs—and now user manuals—are (more or less) the same thing? The arguments that Prolog programs are executable specifications has been well-rehearsed elsewhere (e.g., [4]). Our method would also work with other representations, for example, by representing the interactive system as a finite state machine, which indeed we have already done.

A complete DF200 manual generated by our prototype would run to many pages. We have done sufficient work to show the principles; to convince ourselves and others that the remainder is routine; and to give us a good estimate of how long it would take to implement the remainder. The major effort was spent by the first author in reverse engineering the DF200, primarily because of the arbitrariness of the system design and the difficulties encountered understanding the DF200 accurately enough. The specification of the DF200 is included below as an appendix together with extracts from the manual; the Prolog program itself is about one page of routine programming.

4 Comparison with DF200 Manual

Subject only to the effort of writing a long manual, the DF200's original manual could be reproduced by this method ... with the following caveat.

The DF200 manual distributed with the product contains numerous errors. There are functions the DF200 supports that are not documented, there are documented features the DF200 does not support, and there are features incorrectly documented. These sorts of infelicity would be easy to avoid by using an automated manual generation method such as ours. Or, if preferred, our method could highlight to the technical writer the occasions on which they occur. The programmer, too, might have had an opportunity to fix the problems (if the claims of the manual were deemed more appropriate to meet the requirements than the system itself).

5 Further work

We are also in the course of applying the manual generation method in a different context. An analysis of certain Airbus A320 Flight Crew Operating Manual sections was given in [5]. The A320 is a fly-by-wire aircraft, that is, the airplane is controlled by control logic implemented in computer software and hardware, to which the pilots' commands are one set of inputs. The Flight Crew Operating Manual (FCOM) is the document that pilots of the A320 are required by regulations to know and use when operating the airplane. It is naturally important that the pilots understand the control logic, and therefore that the Flight Crew Operating Manual is as well-written and complete as feasible. It seemed an obvious candidate on which to try our methods.

In [5], the operation of the braking systems as described in the FCOM is analysed, and written as a set of state machine diagrams, each with a low number of states (2 or 3 per machine). Consequent to the analysis, a set of diagrams of more complete state machines was proposed. Some of these state machines have been described using our methods, and we are in the process of comparing the output generated using our method with the original manual descriptions of the state transitions. The fragment we have generated is equivalent to or (we judge) better than the original text. Extending to the rest of the braking system description is routine.

In this case, we started from an already-performed analysis of the FCOM (by the second author). Using our method from scratch would have uncovered the sorts of infelicities reported in [5]—and in a way that would have been obvious early in the design cycle. This justifies our claims that a major

advantage of our method is that it incorporates an analysis of the specification. The analysis takes place while generating an explicit state machine (or set of state machines) to process, and from which to build the manual.

We foresee no technical obstacle to extending the manual writer's aid to more complex examples, the only matter being the tradeoff between the ease of use (for the technical author) against the manual production guarantees required.

We generate the manual from its hypertext specification using a topological sort of the sections: this method is flexible, and obtains results with minimal effort from the technical author. If the aid permitted cycles, the criteria for optimal ordering would be an interesting problem in its own right (see [8] for a discussion, though this paper is unaware of standard algorithms; [13] suggests minimal spanning trees).

6 Conclusion

Using our approach, user manuals for many interactive devices can be generated. They are guaranteed to be complete, and also correct provided that certain underlying algorithms and the original specification are correct, and the technical authors are fluent. Considering how easily we implemented our prototype, it is surprising such automation is not more widely used. We have noted the many and considerable advantages of generating guaranteed manuals from specifications.

References

[1] M. Addison & H. W. Thimbleby (1994), "Manuals as Structured Programs," in G. Cockton, S. W. Draper & G. R. S. Weir (editors), *BCS Conference HCI'94*, People and Computers, **IX**, pp67–79, Cambridge University Press.

[2] BT (undated), *DF200 User guide*, (Apparently coded: UM DF 200 23175349-2), British Telecommunications plc., London.

[3] D. E. Knuth (1984), "Literate Programming," *Computer Journal*, **27**(2), pp97–111.

[4] R. Kowalski (1979), "Algorithm = Logic + Control," *Communications of the ACM*, **22**, pp424–436.

[5] P. B. Ladkin (in press), "Analysis of a Technical Description of the Airbus A320 Braking System," *High Integrity Systems*, **1**(4).

[6] J. Nielsen (1993), *Usability Engineering*, Academic Press.

[7] J. Powers (1994), "Giving Users Help With Apple Guide," *Develop* (The Apple Technical Journal), **18**, pp6–21.

[8] M. Sharples, J. Goodlet & A. Clutterbuck (1994), "A Comparison of Algorithms for Hypertext Notes Network Linearization," *International Journal of Human-Computer Studies*, **40**(4), pp727–752.

[9] H. W. Thimbleby (1986), "Experiences with Literate Programming Using CWEB (A Variant of Knuth's WEB)," *Computer Journal*, **29**(3), pp201–211.

[10] H. W. Thimbleby (1987), "The Design of a Terminal Independent Package," *Software—Practice and Experience*, **17**(15), pp351–367.

[11] H. W. Thimbleby (1989), "A Literate Program for File Comparison," in Literate Programming, *Communications of the ACM*, **32**(6), pp740–755.

[12] H. W. Thimbleby (1990), "You're Right About the Cure: Don't Do That," *Interacting with Computers*, **2**(1), pp8–25.

[13] H. W. Thimbleby (1993), "Combining Systems and Manuals," *Proceedings BCS Conference on Human-Computer Interaction*, **VIII**, HCI'93, Loughborough, J. L. Alty, D. Diaper & S. Guest (editors), pp479–488, Cambridge University Press.

[14] H. W. Thimbleby (1994), "Formulating Usability," *ACM SIGCHI Bulletin*, **26**(2), pp59–64.

[15] L. Wittgenstein (1961), *Tractatus Logico-Philosophicus*, Trans. D. F. Pears & B. F. McGuiness, Routledge.

A The DF200 specification

The DF200 specification we use is taken directly from a reference diagram in the manual [2, p.34], supplemented with the command for copying (which is not summarised in the manual) and simple tags (`topLevel`, `dud`, ...) that clarify the manual's own inaccurate explanation of the diagram. It can be seen that the specification is quite small.

The names (e.g., '`PRESS OPTIONS`') are taken verbatim from the manual; they are not, in fact, exactly what the DF200 displays! Notice that the fax's receive and transmit speed selections are different. (A proper critique of the DF200's user interface *per se* will be the subject of a subsequent paper.)

Like the DF200's manual, we do not provide additional explanation as to what the various named functions *do*; the specification can easily be extended with suitable comments, though the main purpose is to be able to generate explanations of *how* to do the functions—saying what they do can be postponed to the manual writing phase.

```
system(df200, M) :-
   ReceiveSpeeds = ['9600', '4800', '2400'],
   TransmitSpeeds = ['9600', '7200', '4800', '2400'],
   WithWithout = ['WITH', 'WITHOUT'],
   M = tree(topLevel, '(the time)',
    ['Copy',
     tree(digitSelect, 'PRESS OPTIONS',
      [tree('INITIALISATION',
        [tree('PARAMETERS',
                 [tree(startSelect, 'USER PARAM.',
                    ['DATE AND TIME', 'FAX NUMBER', 'ID', 'PASSWORD',
              tree(optionsSelect, 'ACCESS', ['FREE', 'PROTECTED'])]),
          tree(startSelect, 'TRANSMISSION PARAM.',
            [tree(optionsSelect, 'HEADER TRANSMISSION', WithWithout),
             tree(optionsSelect, 'TRANSMISSION REPORT', WithWithout),
             tree(optionsSelect, 'LOCK TRANSMISSION SPEED', TransmitSpeeds)]),
          tree(startSelect, 'RECEPTION PARAM.',
            [tree(optionsSelect, 'RECEPTION MODE',
                   ['AUTOMATIC', 'MANUAL', 'PROGRAMMED']),
             tree(optionsSelect, 'RECEPTION TYPE',
                   ['FAX-ANSWER', 'FAX', 'ANSWER']),
             tree(optionsSelect, 'PRINT HEADER', WithWithout),
             tree(optionsSelect, 'LOCK RECEPTION SPEED', receiveSpeeds),
             'NUMBER OF RINGS']),
                 tree(startSelect, 'NETWORK PARAM.',
            [tree(optionsSelect, 'NETWORK TYPE', ['PUBLIC', 'PRIVATE']),
             tree(optionsSelect, 'DIALLING', ['PULSE', 'TONE']),
             tree(optionsSelect, 'SPEED DISPLAY', WithWithout)]),
                 tree(startSelect, 'ANSWER PARAM.',
                   [tree(optionsSelect, 'ANSWER', ['SIMPLE', 'RECORD']),
             tree(optionsSelect, 'REMOTE INQUIRY', WithWithout),
             'ACCESS CODE',
             tree(startSelect, 'MESSAGE DURATION', WithWithout)])]),
        tree('ONE-TOUCH PHONE', ['ENTER KEY', 'ENTER ID', 'ENTER NUMBER']),
        tree('2-DIGIT ABBR. NUMBER',
           ['ENTER 2-DIGIT ABBR. NUMBER', 'ENTER ID', 'ENTER NUMBER',
            tree(optionsSelect, 'IND. LOCK SPEED', TransmitSpeeds)]),
        tree(startSelect, 'ONE-TOUCH FAX',
```

```
            ['ENTER KEY', 'ENTER ID', 'ENTER NUMBER',
             tree(optionsSelect, 'IND. LOCK SPEED', TransmitSpeeds)])
        ]),
        tree(delayFax, 'DEL. TRANSMISSION',
           ['ENTER PAGES', 'ENTER NUMBER', 'ENTER TIME']),
        tree(setAlarm, 'ALARMS', ['ALARM ON', 'ALARM OFF']),
        tree('POLLING/DEPOSITING', ['POLLING BASE', 'POLLING']),
        tree('PRINTING',
           ['TRANSMISSION LOG', 'RECEPTION LOG', 'ONE-TOUCH FAX',
            'ONE-TOUCH PHONE', '2-DIGIT ABBR. NUMBER', 'PARAMETERS']),
        tree('ANSWERING/RECORD',
           ['ANNOUNCEMENT LISTENING', 'ANNOUNCEMENT RECORDING']),
        tree(dud, 'COMMANDS', ['PRINT', 'DELETE', 'PERFORM'])
])]).
```

The tags are used to generate specific comments for the skeleton manual. (In production work, more detailed comments would be desirable, and they might also be combined directly with the specification—in which case they would also be available to a running program.)

```
comments(dud, '** This feature is not supported!').
comments(delayFax, 'Insert paper first.').
comments(setAlarm,
  'If you just set the alarm on, it rings immediately.
   You must always set a time.').
```

The structural specification above is supplemented with rules to explain the tags as specific sequences of key strokes the user requires to activate various functions.

```
% key*N means press the key N times
keydata(digitSelect(N), digit-NN) :- NN is N+1.
keydata(startSelect(N), 'Start/Copy'*N).
keydata(optionsSelect(N), 'OPTIONS'*N).
keydata(normal(N), ['OPTIONS'*N, 'Start/Copy']).
keydata(topLevel(0), 'Start/Copy').
keydata(topLevel(1), 'OPTIONS').

sameKeys(dud, normal).
sameKeys(delayFax, startSelect).
sameKeys(setAlarm, normal).
```

B Extracts from the skeleton manual

Given the above specification, a short Prolog program generates skeleton explanations. For example, by submitting the Prolog query `writeManual('TONE')?` we obtain (in LaTeX):

To do TONE

Press OPTIONS. Press digit-1. Press Start/Copy. Press OPTIONS 3 times. Press Start/Copy twice. Press OPTIONS.

The LCD display shows successively: (the time), PRESS OPTIONS, INITIALISATION, PARAMETERS, NETWORK PARAM., DIALLING, TONE.

The skeleton for the entire manual is generated by `writeManual(_), fail?`. Here are further examples:

To do AUTOMATIC

Press OPTIONS. Press digit-1. Press Start/Copy. Press OPTIONS twice. Press Start/Copy.

The LCD display shows successively: (the time), PRESS OPTIONS, INITIALISATION, PARAMETERS, RECEPTION PARAM., RECEPTION MODE, AUTOMATIC.

To do ALARM ON

If you just set the alarm on, it rings immediately. You must always set a time. Press OPTIONS. Press digit-3. Press Start/Copy.

The LCD display shows successively: (the time), PRESS OPTIONS, ALARMS, ALARM ON.

To do PRINT

**** This feature is not supported!** Press OPTIONS. Press digit-7. Press Start/Copy.

The LCD display shows successively: (the time), PRESS OPTIONS, COMMANDS, PRINT.

C Extracts from the final manual

Below is an extract from a partially completed user manual for the DF200. Although in the extract, the technical author has not completed checking out the manual, some section headings have already been defined. Notice how the manual is usable, but unfinished parts are appropriately annotated (LaTeX `\marginpar{}` commands, which normally appear distinctively in the margins, are formatted for these conference proceedings as boxed text).

Setup procedures

To do TONE

Press OPTIONS. Press digit-1. Press Start/Copy. Press OPTIONS 3 times. Press Start/Copy twice. Press OPTIONS.

Not checked out.

The LCD display shows successively: (the time), PRESS OPTIONS, INITIALISATION, PARAMETERS, NETWORK PARAM., DIALLING, TONE.

. . .

Using the alarm feature

To set the alarm on

Press OPTIONS, then digit-3, and the LCD will display `ALARM ON`. Set the time you want the alarm to go off, then press Start/Copy.

Checked 12 Jan 95 at 15:30.

Note that if you just set the alarm on, it rings immediately. You must always set a time for it to work.

To set the alarm off

As for setting the alarm on, press OPTIONS then digit-3. Now press OPTIONS. The LCD will show `ALARM OFF`. Press Start/Copy to confirm you want the alarm off.

Checked 11 Jan 95 at 10:30.

Unsupported features

To do PRINT

**** This feature is not supported!** Press OPTIONS. Press digit-7. Press Start/Copy.
Not checked out.

The LCD display shows successively: (the time), PRESS OPTIONS, COMMANDS, PRINT.

Manual report

This version of the manual generated 16:40 on 12 Jan 95.

70 sections remain to be properly documented; 72 sections remain to be checked out. The system specification was last changed on 8 Jan 95 at 14:30. The manual was checked over the period 11 Jan 95 10:30 to 12 Jan 95 15:30.