T H E A N N O T A T E D A S S I S T A N T:

A Step Towards Human Engineering

Andrew Singer*
Henry Ledgard*
Jon Hueras**

TR- 78-10

* Computer and Information Science Dept.
University of Massachusetts, Amherst 01003

** Information and Computer Science Dept.
University of California, Irvine 92717

This work was supported by the National Science Foundation and the U.S. Army Research Office at Durham, N.C.

Alice had been looking over his shoulder with some curiosity. "What a funny watch!"[5] she remarked. "It tells the day of the month, and doesn't tell what o'clock it is!"

"Why should it?" muttered the Hatter. "Does *your* watch tell you what year it is?"

"Of course not," Alice replied very readily: "but that's because it stays the same year for such a long time together."

"Which is just the case with *mine*," said the Hatter.

Alice felt dreadfully puzzled. The Hatter's remark seemed to her to have no sort of meaning in it, and yet it was certainly English. "I don't quite understand you," she said, as politely as she could.

"The Dormouse is asleep again," said the Hatter, and he poured a little hot tea upon its nose.

5. An even funnier watch is the Outlandish Watch owned by the German professor in Chapter 23 of *Sylvie and Bruno*. Setting its hands back in time has the result of setting events themselves back to the time indicated by the hands; an interesting anticipation of H. G. Wells's *The Time Machine*. But that is not all. Pressing a "reversal peg" on the Outlandish Watch starts events moving *backward;* a kind of looking-glass reversal of time's linear dimension.

One is reminded also of an earlier piece by Carroll in which he proves that a stopped clock is more accurate than one that loses a minute a day. The first clock is exactly right twice every twenty-four hours, whereas the other clock is exactly right only once in two years. "You *might* go on to ask," Carroll adds, "'How am I to know when eight o'clock *does* come? My clock will not tell me.' Be patient: you know that when eight o'clock comes your clock is right; very good; then your rule is this: keep your eyes fixed on the clock and the *very moment it is right* it will be eight o'clock."

— THE ANNOTATED ALICE

The Hatter's watch nicely illustrates the effect of idiosyncracy in system design. Really, a watch could provide any number of features, but most watches designed for people put a high priority on telling the correct time of day. Thus, the Hatter's watch is an excellent example of bad human engineering. By human engineering we mean "the selection among design alternatives so as to relate to people." Carroll's stopped watch is the ultimate in poor human engineering because the user must do all the work.

Introduction

Over the years many authors (Cooke and Bunt 1975, Cuandra 1971, Holt and Stevenson 1977, Kennedy 1974, Mann 1975, Palme 1975, Parsons 1970, Sterling 1974, Vandenberg 1967, and Weinberg 1971) have made the case for human engineering of computer systems. As Sterling (1974) and Holt and Stevenson (1977) have pointed out, human engineering is something that must be integrated into the design process, i.e. *it cannot be grafted on later.* We believe that very few systems have been designed with first priority given to human factors. The work described here reflects a conscious attempt to design a computer system in which human considerations had top priority in the design process.

As part of our general attempt to limit the influence of implementation considerations, at the start we chose to complete the design of every system feature before undertaking any implementation. Thus far we have adhered to that decision, and at present we are working on a detailed formal description of the entire system.

For a variety of reasons, we undertook the task of developing a standard interactive environment for PASCAL programmers, although the use of PASCAL is incidental to our design. We were interested in assisting both naive and sophisticated programmers.

The effort was motivated by several concerns:

1. the development of a moderately powerful system that makes users more productive with less effort,
2. the need for a system that stimulates rather than dampens the enthusiasm of potential users,
3. a desire to create a system without the trappings of conventional computer concepts, terminology, and jargon,
4. the need for documentation embodying a light and friendly approach to users.

These goals are easy to talk about, but difficult to realize. After nearly two years' work, we produced a design carefully documented in the form of a User's Guide. This document itself was the result of considerable effort.

The final design represents a large number of rewrites, perhaps ten, aimed at making the system more accessible to the user. We consider this document to be an example of our overall concern with the human engineering of the system as a whole.

In this paper, we present the User's Guide (with one appendix) in annotated form. The notes are intended to illuminate the human enginnering design considerations and to explain the principles motivating our decisions.

It is important to bear in mind that the state of our knowledge about what constitutes a "good" decision from a human factors point of view is rather primitive. Often our only guide is intuition based on experience. Ultimately, we believe that the results of existing experimental investigations is psychology, new human engineering experiments in computing will provide a solid basis for designs. Nevertheless, even without much data we must give current systems the benefit of the best knowledge that we have. This is the case here.

In a sense, whenever we build a system today we conduct a human factors experiment. Unfortunately, we are rarely in a position to extract any valuable data from these experiments because we do not set them up as such. In the design of the Assistant, we developed a number of general principles to guide our decisions. As people use the system we expect to discover that some of these are wrong. But at the very least, we know we will be able to learn from our experience. Given the limits of hard data, we believe that work must proceed in this way if we are to advance the state of human engineering in computer systems.

The Current Status of the Assistant

The design described here was frozen in the summer of 1976. Since then we have been engaged in a variety of tasks related to it. Considerable effort has gone into a full formal definition of the Assistant. At present, a complete definition of its interactive behavior is done, and a partial definition of the semantics of data manipulation performed by the Assistant's editing requests has been written. A text editor, HOPE, based on the Assistant' editing request has been written in PASCAL and will soon be available from the PASCAL Users Group. The stand-alone automatic prettyprinting program (also in PASCAL) mentioned in the notes has been available from the University of Massachusetts for the last year and a half and currently has been distributed to more than seventy installations. Continuing efforts are aimed at completing the formal definition of the Assistant and then building a prototype. All of these efforts, notably the design of the Assistant and its human factors considerations, have required about ten person-years on effort, which we consider as research.

In reading this paper, it is important to bear in mind that what we are discussing is the *design*, not the implementation, of a system that we believe will be well within the state of the art. Until the complete formal description or an actual implementation has been completed, it remains to be seen whether this is the case.

All of this activity has not been without its effects. Two recent papers (Singer and Ledgard 1977, Ledgard and Singer 1977) describe some of our emerging beliefs. A number of specific second thoughts concerning the design of the Assistant have also emerged. Whether we will incorporate these into a Revised User's Guide remains to be seen; but for the sake of completeness, we have included these thoughts in the following annotations.

A USER'S GUIDE to THE PASCAL ASSISTANT

Introduction

" assistant ... 1. one who assists or gives aid and support; a helper; ... "

- Random House Dictionary of the English Language

" automation ... 1. a mechanism that is relatively self-operating; esp. ROBOT 2. a machine or control mechanism designed to follow automatically a predetermined sequence of operations or respond to encoded instructions 3. a creature who acts in a mechanical fashion ... "

- Webster's New Collegiate Dictionary

The Assistant[1] This section is your introduction to a little robot we have created called "The PASCAL Assistant." This Assistant can help you create, manipulate, and execute PASCAL programs. Like any creature, natural or artificial, the Assistant has its own ideas about things. Unlike ourselves, however, the Assistant's ideas are fairly fixed, and its intelligence is limited. As with any assistant, your understanding of it will make for a smooth working relationship.[2]

Terminals The Assistant exists as a collection of computer programs that run on a time-sharing computer. Since you and the Assistant interact solely by means of a teletype or some other interactive terminal, we will at times describe the Assistant's behavior in terms of what the terminal does. Because you may be using any one of a variety of terminal devices, we can only describe what happens in a general way. For specific details concerning individual terminals, we refer you to "Appendix 1: Sign-on Procedures and Terminals."[3]

Goals The Assistant's aim is to function in a way that will be pleasant and helpful to you. In this end, the Assistant follows three general strategies:

1. An important part of the human engineering of a system is the physical display and organization of its documents. Throughout the User's Guide we attempt to keep a layout that is both visually appealing and yet can be used for quick reference. Because the User's Guide is short, there is little need for an index. Instead, keywords and key phrases are given in the left margin of the manual. (In the published version of the User's Guide, they occupy a separate column.) These keywords also give the reader a quick clue about the basic idea being presented. We are indebted to Child, Bertholle, and Beck (1971) for this effective scheme.
Perhaps of most importance, the scale of the manual is small. Of course, this is reflected in the smallness of the design itself. Nevertheless, a great deal of care was exercised in eliminating details that the user should, in fact, find out for himself on the system. This is not to say that we believe the manual is incomplete or misleading; rather a great effort was made to present the system in as concise a manner as possible.

2. One of the most controversial choices we made was to present the Assistant to the user in a consciously anthropomorphic form. From the beginning we describe the Assistant as a creature, robot-like, with a goal structure, consistent behavioral rules, interactive strategies and deductive capabilities. This idea was motivated by several considerations.

In the course of a terminal session, the user must keep track of a great deal of information. For example, the user must continually be aware of the status of his files, his current level of interaction with the system, the consequences of actions he has already taken, and the actions he may legitimately take next. Furthermore, since the actions he may perform are primitive, he must repeatedly supply redundant information over a long sequence of requests. Finally, he must constantly be on guard against destroying his own work by doing something that might seem innocuous, but results in disaster.

In most cases, the kind of information at issue here is readily available to the system. It was our intent that the Assistant take full advantage of the knowledge availabl- to it, and relieve the user of much of the burden of constantly juggling that knowledge.

When we tried to describe a primitive knowledge-based system in a way that would be simple and non-threatening to users, the natural step was to deliberately exploit the creature-like view which people inevitably apply to machines anyway.

However, what began as a conceptual model for the documentation quickly acquired a life of its own and repeatedly suggested consistent directions for various aspects of the design. This interaction between the documentation and the design was not limited to the conceptual model of the Assistant. In general, whenever we found a feature or concept difficult to explain clearly, we took this as a signal that the design itself was likely at fault. Whole versions of the editing requests were rejected because we could discover no simple way of explaining them.

We benefited in other ways from this view of the Assistant as a creature. For one thing, we were able to avoid much of the dryness associated with the normal type of manual. And, as the opening paragraph suggests, we were able to introduce a light touch for the reader, and thus make use of the guide a more pleasant experience.

3. Only one of the three appendices to the User's Guide is presented in this paper. Nevertheless, it is important to note that the complete User's Guide (including all the appendices) comprises a document that contains everything the user has to know about the system. The reader need refer to no other documents. This is consistent with recommendations made by Vandenberg (1967).

4. One of the advantages of an interactive system for users is that interaction can be used to couple the system and the user. Unfortunately, few systems provide more than negative feedback to a user, i.e. error messages. Positive reinforcement in the form of highly specific confirmatory messages or an ongoing "chatter" from the system can simultaneously teach the new user what to expect and reassure the user that what he expects is, in fact, going on. A wide body of evidence is psychological reinforcement theory supports the value of this strategy. A further benefit of this interactive strategy is that it allows the resolution of potential ambiguities that may arise in a request. Accommodating such ambiguities permits more flexibility in the language design, especially as regards abbreviated forms.

1. It provides you with continuous information about its activity.[4]
2. It makes reasonable assumptions about what you want to be done when specific details are not given.[5]
3. It checks with you before carrying out a potential damaging operation.[6]

Interaction with the Assistant

These strategies imply a large amount of interaction between you and the Assistant, especially when you call on it for the first time. However, you will find that interacting by means of a terminal can become tedious, particularly if the terminal is slow or if both you and the Assistant are capable of working at a much quicker pace. As you become more familiar with the Assistant, you may direct it to assume that your interaction is to be more abbreviated, just as you may at any time direct it to assume certain other things about your working environment.[7]

Request Language

Requests are made to the Assistant via the terminal and are expressed in terms of a "request language." This language is designed to look very much like English and consists entirely of imperative statements. Several requests allow you to exchange information with the Assistant concerning almost anything within the scope of its knowledge.

Behavior

At certain times, the Assistant may be attentive, which means that it is awaiting a response from you. At other times, the Assistant may be active, which means that it is trying to satisfy a request for you. Sometimes before a request is satisfied, the Assistant discovers that it needs more information from you, in which case it will ask you what it needs to know.

Attentive Behavior

Attentive behavior is always signaled by a prompting message, which consists of two characters typed by the Assistant on the terminal. The prompting message indicates not only that the Assistant is awaiting a response from you, but also what type of response is being asked for.

5. As Gilb and Weinberg (1977) point out, extensive use of "natural" defaults is inherent in all natural language communication, and such defaults may be explicit or implicit. The Assistant is designed to take advantage of both types.
For example, if a program is to be run and no compiled version of it is handy, the Assistant implicitly assumes that it must first be compiled. Moreover, at any time the user may explicitly direct the Assistant to make explicit assumptions about future requests. Thus, the user does not have to continue to specify file names, line boundaries, or options for requests when the system can keep track of these details.

6. The philosophy of "security checking" is not novel, but is also not commonplace, and the extent to which it is used by the Assistant may seem extreme. A frequent example is an attempt to overwrite files. Unless told otherwise, the Assistant will always inform the user that a file is about to be destroyed, ask for confirmation, and thus give the user a chance to think twice before going ahead. Another, less obvious, example is the warning a user receives when a text deletion request threatens to destroy a large part of a file. These security checks are intended to give the user confidence that the system will warn him before doing something disastrous. A skilled user may suppress most of these checks.

7. Ideally, we would like an Assistant that knows what level of detail the user needs and adapts automatically; but such intelligence is beyond the limits of cost effectiveness that we have set. The Assistant is intended to be semi-intelligent, only an incremental step toward a truly intelligent system. What we cannot accomplish with limited intelligence we have tried to accommodate with a user-driven adaptive strategy. In some ways, this is one of the least satisfying approaches that we have employed in the Assistant. Not surprisingly, the complexity of the ASSUME request, our vehicle for adaptation, reflects this.

Active Behavior

When you send the Assistant a request, it becomes active and attempts to satisfy your request. It does this in three stages:[8]

1. Verification - The Assistant determines whether or not your request makes sense, and makes any necessary assumptions that it can when specific details are not given.
2. Performance - If the verification stage was completed successfully, the Assistant will satisfy your request. If the operation requested is at all time-consuming, the Assistant may indicate its progress at various intervals.
3. Completion - After your request has been satisfied, the Assistant indicates the final result of its actions and again becomes attentive.

8. There is a body of psychological evidence (see, for example, Thorndike and Rock, 1934) which suggests that people "learn without awareness." One implication of these results is that the users of a computer system will infer underlying principles even if they are unaware of doing so.

The Assistant's behavioral goals are not merely "sugaring", but are accurately reflected in its responses. These goals are intended to help the user make reasonable inferences about what the Assistant will do with a particular request. For example, the first goal, verification, ensures that no request will be executed unless it makes sense semantically. In some cases, this implies that significant static prechecking must be performed. This seems a small price to pay for relieving the user of the burden of correcting damage done by a technically legal but senseless request.

Interrupting the Assistant

While the Assistant is active, you may interrupt it at any time, causing it to become attentive again. You interrupt the Assistant in order to ask it for pertinent information or else to tell it to discontinue attempting to satisfy a request for you and to attempt to satisfy a new one.[9]

9. Most interactive systems have some form of interrupt, but like other details, the interpretation placed on it is often inconsistent or counter-intuitive. The Assistant's interrupt is like a tap on the shoulder. Following an interruption, the Assistant suspends what it is doing, returns with an explanation about what is going on, and asks the user whether he wishes to continue the task. At this point, the user may reply or request additional information. If the user does request additional information, this request may itself be interrupted, but such interruption simply terminates the request for additional information and returns to the original level of interruption. Again, the user is reminded about his original interruption and is asked what to do. Thus, there is no confusing "stacking" of interrupts as, for example, in APL (Wiedmann, 1974), but interruption is always a possible and meaningful operation. This possibility of interrupting a task and carrying out a dialogue concerning the task is patterned after normal discourse. (See Mann, 1976 and Palme, 1975). From our view, it is one of the cleanest features to emerge in our design.

Error Conditions

There are several conditions under which a request cannot be satisfied:

1. If the request cannot be understood or is inconsistent with what is known,
2. if the Assistant asks you to confirm a request and you do not comply, or
3. if the performance of the request fails for some reason.

Immediate Error Correction

When a request cannot be satisfied, the Assistant will identify the problem and become attentive. If an error is found in the verification stage, no action will be taken. At this point, you may easily modify and reissue your request using a request correction facility. Alternately, you may issue an entirely new request.[10]

<u>Files</u> A file is a named collection of information that the Assistant maintains for you. The Assistant's primary function is to provide you with a means of creating, manipulating, and performing various operations with files. Most files that you will use will be files of text, and many of these will be PASCAL programs. When you create a file, you give it a name. From then on, both you and the Assistant refer to that file by the file's name.[11]

<u>Preserving Files</u> Any file that you create during a session with the Assistant will be kept for the duration of the session. It may be kept for future sessions provided that you specifically ask the Assistant to preserve it for you. No file will be discarded without your prior approval. Files previously preserved can be modified at any time. However, at some point the Assistant must be told whether or not these modifications are to be preserved as well. Once a modified file has been preserved, its previous condition is lost forever.[12]

<u>Assumptions</u> The Assistant retains information about what you have done and what you have explicitly asked it to assume. Initially, the Assistant uses some basic assumptions about how you, as a beginner, would want it to behave.[13] Assumptions, as we've said, enable the Assistant to reach reasonable conclusions about what you want done when certain details are omitted from a request. Thus, the use of assumptions frees you from having to supply excruciating amounts of detail.

<u>Limitations</u> As we stated earlier, the Assistant has a very limited understanding. It can make only very simple deductions based on its restricted knowledge. When you try to give it a request that it does not understand, it will tell you so, but it cannot really inform you of the limits of its own intelligence. This does not mean that its intelligence is illusory. In fact, you may very well find its perceptiveness surprising at times.[14]

10. It is a fundamental premise in the Assistant's design that users will make errors. Many interactive systems have facilities for deleting characters or lines as they are being entered. Unfortunately, a user may discover such an error well after it is made. The immediate correction feature is designed to make it simple for users to correct such errors quickly, and without retyping the entire line. If the user makes an error and, as a result, the Assistant discovers that a request is ill-formed, the Assistant will report the error. The user may then change the erroneous line with a conventional edit request, and the Assistant will automatically re-issue the corrected request. While we have never seen this simple feature elsewhere, we believe that it is especially useful for lengthy editing requests and multiple request lines where typing errors are particularly frustrating.

In a similar vein, the UNDO request is a means of erasing the effect of a request that was performed but did not produce the result desired by the user. The UNDO feature will likely be limited to editing and assume requests where implementation will not cause severe difficulties.

These are both examples of the way the Assistant keeps track of things; in this case, an immediately preceding but unsatisfactory request.

11. While a serious attempt was made in the Assistant's design to avoid the terminology of conventional systems, the concept of a file seemed inevitable. In a private correspondence, Hoare (1976) suggested the alternative notion of "books" or "folders" supported by an appropriate graphic display. Our decision to support printing terminals ruled this out.

12. The Assistant uses a simple two-level file system. A good deal of effort went into designing this system so that its operation is largely automatic and transparent to the user. When the user directly refers to a new file, a current temporary copy of it is created. All operations are performed on the current version. At the end of an interactive session, the Assistant asks the user what to do with files that do not have equivalent permanent copies. Although the user must be aware that, potentially, there are two copies of his file, the management

Some Notation[15]

Grammatical Notation Every language has a grammar, and the Assistant's request language is no exception. Because the Assistant identifies your requests by their form, grammar is especially important in communicating with it. In the descriptions that follow, we employ a special notation to describe the grammatical form of each request. The rules for this notation are as follows:

Keywords

1. Words shown in upper case are *keywords*. Keywords are like guideposts to the Assistant. They signal what to do and what to expect. Except for PRESERVE, RESTORE, and DESTORY, any keyword may be abbreviated by its first letter. If not abbreviated, a keyword must be spelled out correctly. (See **17**.)

Objects

2. Words shown in lower case and connected with hyphens ("-") are names for the objects of a request that you supply to make the request specific, such as the name of a file, a mode of interaction or a piece of text.

Alternatives

3. Keywords or objects that are grouped together and separated by slashes ("/") are mutually exclusive alternatives. For example:

   n/ALL

   means that either "n" (a number) or the keyword "ALL" may be specified, but not both.

Options

4. Keywords, objects, or any groupings of these that are in parentheses represent parts of a request whose use is optional. For example:

   QUIT (QUICKLY)

   means that you may say "QUIT QUICKLY" or simply "QUIT."

Ordering

5. Keywords, objects, or groupings of these may only be specified in the order in which they appear in a rule.

of these files is left largely to the Assistant. Specific file manipulation requests enable a user to preserve the current version of a file or restore it to its previously preserved condition.

In retrospect, it is somewhat surprising how much time we spent designing this scheme. Yet, we believe that the concepts of file restoration and preservation in the Assistant are unusually simple.

13. The assumptions for beginners take nothing for granted and attempt to assure that no beginner will be lost too easily.

14. We had reservations about presenting the Assistant as a semi-intelligent creature with moderate self-consciousness that understands a narrow natural-language subset. There is always the danger that naive users will come to expect too much and thus be frustrated. We have tried to compensate for this by emphasizing limits, but it may not be sufficient. Nevertheless, we feel that the benefits of an approachable conceptual framework for people are significantly greater than the problems it may create.

15. Despite our desire to keep notation and terminology to a minimum, we felt compelled to resort to a kind of context-free grammar. Notations, even simple context-free grammars, can at first be difficult for many users. We attempt a gentle introduction to the use of a few grammatical notations. It is likely that this complexity of the documentation reveals what is probably a weakness in design.

Request Language Summary[16, 17]

General Requests

EXPLAIN	(name)	
SHOW	(name)	
ASSUME	assumption	
GRIPE		
UNDO		
QUIT	(QUICKLY)	

Editing Requests

NEXT	(lines-of-text)	
PREVIOUS	(lines-of-text)	
LIST	(lines-of-text)	
DELETE	(lines-of-text)	
TRANSFER	(lines-of-text)	INTO file[18]
COPY	(lines-of-text)	INTO file
INSERT	(new-lines-of-text)	(BEFORE/AFTER/OVER)
MAKE	text new-text	

File Requests

PRESERVE	(file-text)
RESTORE	(file-text)
DESTROY	(file-text)

Program Requests

RUN	(file-list)	(WITH parameter-file-list)
VERIFY	(file)	(INTO file)
FORMAT	(file)	(INTO file)
BIND	(file-list)	INTO file

16. A major concern in the design was to limit the scale of the Assistant. This was one of the most difficult issues to confront. The tendency to expand and enlarge, to add "powerful" and "important" features was overwhelming. As Miller (1956) pointed out in a stimulating but inconclusive paper, "The Magical Number Seven Plus or Minus Two", there definitely seem to be small limits on our capacity for dealing with large numbers of conceptual objects, but these limits are extended by a phenomenon known as "chunking", in which aggregates can be formed. In spite of the chunking phenomenon, we believe there is a strong intuitive case to be made for keeping things small.

A common criticism voiced over the Assistant's design is that it is a "toy." This was certainly not our intent. However, we have rigorously excluded any feature which we felt would be of use to only a small fraction of users. We believe that the Assistant is an uncommonly simple solution to providing a pleasant and productive working environment for a majority of programmers.

From the request language summary the small scale and symmetry of the Assistant are immediately apparent. What is not so apparent is the capability that lies within this simplicity. Users of HOPE, our prototype of the Assistant's editing requests, have been surprised by the power of what they took to be a fairly simple-minded editor.

17. Another major design decision we made was to base the Assistant's request language on a limited English phrase structure. There were a number of reasons for this choice. The natural language of interaction between people is natural language. Even individuals exceptionally experienced with notation have still greater training in natural language. Thus, our aim was to exploit this natural language experience.

Because a reasonable body of experimental data (see, for example, Weist and Dolezal 1972, Epstein and Arlinsky 1965) suggests that people have difficulty in manipulating language-like formation that violates normal syntactic structure, we tried to follow normal syntax as closely as; and we tried to choose the shortest, most apt, and most orthogonal set of keywords. Short words were chosen not out of typing considerations, but because they occur more frequently and are easier to recall.

While we tried to copy English grammar closely, we did not allow the meaningful reordering of phrases permitted in English, e.g. "Into A, copy B." We avoided this because of the ambiguities it might introduce into the request language, especially in its abbreviated form. It seems more desirable now to use a more relaxed syntax and resolve ambiguities with an interactive exchange with the user.

Seemingly at odds with the decision to follow natural language syntax strictly was the requirement that the request language have an effective abbreviated form. The ideal, of course, would be to have special function keys for each word, but the real world of ordinary terminals precludes that.

The solution was to introduce the uniform abbreviation rule that *any* keyword can be abbreviated by its *first* letter. Furthermore, abbreviated keyword sequences can be typed *without* intervening spaces. These two rules result in an abbreviated form of requests that is fast and easy to type. Because the rule is so simple, the user can think in the long form while typing its abbreviation. (Because of their potential danger to the user, the three file requests were excluded from this general rule and cannot be abbreviated. This now seems paradoxically inconsistent.)

A variety of data suggest the first letter abbreviation rule. A paper by Freedman and Landuaer (1966) points to the usefulness of the initial letter as a recall clue, and some recent work by Selvin Chen-Chance (197-) indicates the importance of the initial letter (for adults at least) as a discrimination cue.

This approach to abbreviations is not a "minor" issue. One of the least thought out philosophies of almost every system we have seen is its abbreviation strategy. Abbreviations, like other so-called "details" of design are often very critical, for such details may be the most frequently encountered features of a system. From the user's point of view, ours is a powerful convention. From a designer's point of view, this convention was almost impossible to live with. On many nights we took a thesaurus to bed.

An argument commonly advanced against our abbreviation rule has been that we could not easily expand the keyword list, i.e. add new requests. In rebuttal, we suggest that such additions would be best accomplished by a complete redesign, if all the interlocking design aspects are to receive the consideration they deserve. Furthermore, as the ASSUME demonstrates, any keywords within a request are free from conflict from keywords within other requests.

18. The TRANSFER and COPY requests are good examples of our attempts to follow a limited English phrase structure. Rather than use conventional notations like "TRANSFER, lines-of-text, file", we borrow from natural English phrase structure.

Unfortunately, we were not completely successful in following English grammar. From a grammatical point of view, the MAKE request would be better as "CHANGE text TO new-text." However, this suffers from the defect of requiring two levels of delimiting, the string delimiters that bracket text, and the syntactical delimiter "TO." Of course, all of this results from the clash between notation (string delimiters) and natural language, an impossible dilemma.

General Requests

The information requests EXPLAIN, SHOW, and ASSUME provide you with the means of exchanging information with the Assistant. You may direct the Assistant to make assumptions about your environment or you may ask it for information about current assumptions, requests, the request grammar, and so on. The use of these requests should make it unnecessary to refer to this User's Guide while interacting with the Assistant.

Explaining Concepts

The EXPLAIN request is your means of getting general information in order to understand something about the Assistant that is not clear. In order to ask about something just say:

EXPLAIN (name)

The "name" you give can be any one of a number of words associated with the request language, error conditions, the Assistant itself, or various concepts behind it, like assumptions or files.

If you say EXPLAIN, omitting any name, the Assistant will respond by giving you information concerning the last thing that you have done or that has happened to you. Each time you say "EXPLAIN" the Assistant will provide you with more information concerning the topic at hand. In addition to its explanation of the given topic, the Assistant may refer you to other related topics.

If the Assistant does not have information on a given name, it will tell you so. If all its information is exhausted, the Assistant will, if possible, suggest external sources (consultants, references, etc.) that you might seek out.[19]

Getting Examples or Specific Data

When you want examples of the request language or specific data concerning files or your working environment, say:

SHOW (name)[20]

In addition to the normal names of things you might ask about, there are several words which will direct the Assistant to show you some special things. These are:

TIME	- The current time of day.

19. A number of interactive systems now incorporate on-line assistance features (e.g. see Teitelman 1974). To the best of our knowledge some of these are integrated into the system so as to take advantage of an awareness of what is going on. The idea of an integrated assistant feature follows naturally from the general interactive strategy of the Assistant and, as such, is simply a request from the user for greater amplification.

The benefits of this approach are several. The user can directly get information that in a conventional system would only be available in a reference manual. Furthermore, this information can be specialized to his situation. Finally, this information is provided in the context of an actual circumstance where its teaching value and reinforcement potential is greatest. (See Bryan and Regan 1972.)

20. The SHOW request is also meant to provide pedagogical examples of the request language. For example, if the user types "SHOW MAKE", the Assistant will give examples of the use of the MAKE request. For both the EXPLAIN and SHOW requests, it is expected that over time more information will be added to the Assistant's knowledge base. Coupled with the GRIPE request, this seems to be a viable approach for improving the Assistant's behavior as our knowledge of what needs explanation expands.

ASSUMPTION	- All of your current assumptions.
FILES	- The names and information concerning your currently preserved files.

Giving Assumptions

In order to tell the Assistant to make specific assumptions about your environment say:

```
ASSUME assumption
```

Assumptions fall into several categories. You can specify one of two modes of interaction by saying:

```
ASSUME INTERACTION IS TERSE/LONG
```
[21]

These two modes are interpreted as follows:

TERSE	- Gives highly abbreviated messages or none at all. Intended for the hotshot user.
LONG	- Gives loquacious messages, spelling everything out from A to Z. Intended for the naive or inexperienced user.

Another category determines the amount of interaction you want the Assistant to assume regarding security checks for potentially dangerous operations. You can specify how much security you want by saying:

```
ASSUME SECURITY IS CAUTIOUS/RISKY
```
[22]

Other uses of the ASSUME request are given further on.

Complaints, Comments, and Suggestions

The Assistant, via EXPLAIN and SHOW, is designed to help you as much as possible within its limited knowledge. However, sometimes this is not enough. You cannot really tell the Assistant your problems and get any kind of sympathy or advice from it. You can, however, tell the people in charge your problems through the Assistant by saying:

```
GRIPE
```

The Assistant will then go into a special attentive mode where you may type in a message of any number of lines. You leave this special mode of interaction by interrupting the Assistant and making a new request. The text you type will be stored, and at regular intervals all the

21. Our original design was based on three modes of interaction, TERSE, MODERATE, and LONG. We are grateful to Hoare (1976) for pointing out that with a good implementation of the EXPLAIN request two modes should be sufficient.

22. As Gilb and Weinberg (1977) point out, at times and for some users, automatic protection and forced interaction may be a nuisance.

messages sent by you and others will be sifted out and examined by the people responsible for maintaining the Assistant.[23]

23. The importance of long range user feedback in maintaining a system cannot be underestimated. In providing a specific request for this, we emphasize its importance and make spontaneous complaints possible. Furthermore, we can take immediate advantage of the system itself to capture inside information about the current state of affairs, which may help us in interpreting a user's complaint.

<u>One Last Chance</u>

If you make a request and you wish you hadn't, you may undo the effect of that request by saying:

```
UNDO
```

The effects of the most recent request made are cancelled, and you may then proceed as if nothing had ever happened.

<u>Leaving the Assistant</u>

In order to dismiss the Assistant say:

```
QUIT (QUICKLY)
```

Before the Assistant will let you go, it will tell you what files have been created or changed and are still to be preserved, and ask you which of those you wish to keep. Furthermore, it will ask you whether or not you want to preserve any new assumptions that you have given it. Finally, it will make doubly sure that you wish to leave before it will let you go.

If you add "QUICKLY" to the request, it will assume that you have already preserved everything you want to keep and will let you go without any fuss.[24]

24. The QUIT request is a good example of our desire to make reasonable and safe assumptions about the user's behavior and still allow more skilled users to override these assumptions.

Editing Requests

Text editing is a process of creating, maintaining, and updating files of text (such as programs, data files, chain letters, or what have you). The Assistant's editing requests make it possible to insert, delete, and substitute text to change the layout and spacing of text, and even to move blocks of text from one file to another.[25]

Editing text commonly requires that a number of changes be made to a particular file. Rather than repeatedly specifying the file to be edited in each request, the Assistant always assumes you want to edit the currently assumed file. (For an explanation of the "currently assumed file" and how it works, see "File Requests - File Assumptions.")

The Current Line

In making editing requests, you must always have some means of specifying what it is you want changed. The Assistant always assumes that a request is made relative to a "current line." Initially, the current line is the first line of the file. Thereafter, each request that references specific lines causes the last line referenced to become the new current line.[26]

Specifying Text

Editing may be performed on whole lines or on pieces of text within a line.

Operations on whose lines may be specified by giving the number of lines from the current line or by giving a piece of text which appears on a line. References to pieces of text require a special notation to describe the text. This notation has the form:

```
=text=
```

The given "text" is any actual sequence of characters. The symbol "=" represents any special character which is neither a letter, digit, space, or semicolon (";"). This special character is used to "bracket" the actual character sequence. Since this character indicates both the beginning and ending of the desired text, it must be a character which does not appear in the text itself.[27]

An example editing session is given at the end of this section.

Moving Forward

To move the current line forward say:

```
NEXT (lines-of-text)
```

There are several ways of describing how many lines of text to advance. The NEXT request

25. In most systems, editing must take place in a special mode or environment. These systems require users to shift levels. The requirement that editing languages be terse usually conflicts with the large scale of the rest of the system. A special editing environment is the logical, if cumbersome, solution to this problem. Then again, many editors are built as independent subsystems and only later incorporated into the main system.

Various studies (e.g. see Turner 1974, Bois 1974) have shown that editing usually accounts for better than fifty percent of the average interactive system's work. Furthermore, the nature of the program development process often leads a user to ping pong between editing and other tasks.

For these reasons, we believe that a text editor must be designed to be an *integral* part of an interactive programming environment. Central to this belief is our feeling that a user should have access to all the capabilities of the system while editing and vice versa. The use of the "assumed" or default file together with the small scale of the Assistant enable us to keep a single-level system for all requests. We are indebted to David Stemple (1975) for making the strong case for this.

26. One of the larger and more difficult decisions we made was to orient the editing requests of the Assistant around the concept of a "current line." Some editors are "character based" (that is, the user's position in the file may occur in the middle of a line), and others are page oriented (i.e. the interaction is always in terms of multiple lines of text).

Obviously, the kind of terminals in use and the kind of text to be edited enter into this decision. We made a deliberate design decision to orient the Assistant around moderate speed (10-30 cps), typewriter-based terminals without a graphic display facility, as these are at present the most commonly used. We also concentrated primarily on the problem of editing programs. While we did not rule out the possibility that the editor might be used for ordinary language text, the special problems of editing such text were not addressed. (See Lance Miller 1977.)

It might have been better to design the Assistant for a more has the following variations:

1. NEXT
   The Assistant moves the current line forward one line.
2. NEXT n
   The Assistant moves the current line forward n (where n is a number) lines.
3. NEXT ALL
   The Assistant moves the current line forward to the *last* line in the file.
4. NEXT =text=
   The Assistant moves the current line forward to the next line containing an occurrence of the specified text.
5. NEXT n =text=
   The Assistant moves the current line forward to the "n-th" line containing an occurrence of the specified text.[28]
6. NEXT ALL =text=
   The Assistant moves the current line forward to the *last* line containing an occurrence of the specified text.

In all of the editing requests, "lines-of-text" has the same general form as shown above.[29]

<u>Moving Backward</u> To move the current line backward say:

PREVIOUS (lines-of-text)

The PREVIOUS request is exactly the reverse of the NEXT request. Note that PREVIOUS ALL takes you to the first line in the file.

<u>Displaying Lines</u> To display one or more lines of text just say:

LIST (lines-of-text)

The variations on the LIST request are similar to the NEXT and PREVIOUS requests:

1. LIST
   Only the current line is displayed on the terminal.

advanced type of high speed terminal. Indeed, with a bit more storage, some of the "intelligent" terminals made possible by recent advances in semi-conductor technology seem entirely capable of supporting an Assistant locally. The parallelism of display, cursor facilities, definable function keys, and the fast display rate afforded by such terminals would make possible substantial improvements in the design of the Assistant, particularly with regard to the editing requests and the management of defaults.

In our opinion, most editors based on the "current line" concept suffer from the drawback that the user must mentally keep track of what the current line is. This defect results from an inconsistent strategy with respect to line pointer movement. In the Assistant we have deliberately avoided this possible confusion.

The current line is *always* the last line seem by the user. The advantage of this strategy is that the terminal is always displaying the current line. The disadvantage is that the examination of text may force an extra step, i.e. moving back to the beginning of text which is to be displayed. Clearly, there are arguments on both sides. Here again, we believe that the value of the general rule outweighs the merits of a special case. Certainly, this issue deserves some thoughtful experiments.

27. Here again, our use of special notation reveals a weakness of design. We remain dissatisfied with this, but find other alternatives even less attractive.

28. A particularly sticky, but important, detail. Should it be the *n*-th o-currence or *n*-th line containing an occurrence? The former seems right for a character-oriented editor, while the latter seems more suited to our line-oriented editor. Endless hours were spent on this issue, with no clear resolution.

2. LIST n/ALL
The Assistant displays the next n (or ALL) lines including the current line.

3. LIST n/ALL =text=
The next n (or ALL) lines containing the specified text are displayed.

29. Getting all the editing requests to conform to the same general format for target text patterns was the result of great attention to detail and numerous debates about the proper function of requests. In doing so, we significantly reduced the amount of information a user must learn and remember.

<u>Deleting Lines</u> In order to delete one or more lines of text you say:

```
DELETE (lines-of-text)
```

This operation is virtually identical to the LIST request with the difference that the particular lines specified are not displayed but *removed* from the assumed file.[30]

30. An intended security check confirms major deletions with the user.

<u>Moving Lines</u> To move one or more lines of text out of the assumed file and into another file say:

```
TRANSFER (lines-of-text) INTO file
```

This request removes the lines of text you specify from the assumed file and puts them into the other file that you name. The lines that are removed will replace the previous contents of the file.[31]

31. It is not obvious from the User's Guide, but the TRANSFER request is not only intended to excise lines from a file but is the basic mechanism for moving blocks of text within a file. By transfering lines of text to a temporary file, the user can later insert the lines at another point in the file using an INSERT request. This two-step process seemed to offer the user a great sense of security for an operation which on a typewriter-like terminal cannot be visualized very well.

<u>Duplicating Lines</u> To make a copy of one or more lines of text from the assumed file and place them in another file say:

```
COPY (lines-of-text) INTO file
```

This request is exactly like the TRANSFER request except that no lines are removed from the assumed file. Instead, copies of the specified lines of text are placed in the named file.

<u>Inserting Lines</u> To insert new lines of text into the CURRENTFILE say:

```
INSERT (new-lines-of-text) (BEFORE/
                          AFTER/OVER)
```
[32]

32. A typical question in many text editors is whether to insert new text before or after the current line (or character) position. The problem is especially troublesome at the beginning and end of a file. The Assistant takes the view that the user should be able to do either, as well as to be able to insert one or more lines of text in place of the current line. This eliminates the confusion associated with inserting text at the beginning or end of a file, without requiring the user to be aware of an imaginary line located at the end of a file or before the beginning of a file.

The variations on the INSERT request are as follows:

1. INSERT (BEFORE/AFTER/OVER)
The Assistant will continually prompt you for lines of input from the terminal until you interrupt the Assistant. The lines you type will be inserted

before, after, or instead of the current line.[33]

2. `INSERT =text= (BEFORE/AFTER/OVER)`

   The Assistant will insert the lines specified by text before, after, or instead of the current line.[34]

3. `INSERT file (BEFORE/AFTER/OVER)`

   The Assistant will insert the contents of the named file before, after, or instead of the current line.

If BEFORE, AFTER, or OVER is not specified, AFTER is assumed.

33. The use of the interrupt to terminate continuous text input is consistent with the general semantics of interrupts and allows for easy input of empty (blank) lines, by far the most frequently entered line of text. Empty lines can be entered by simply typing a carriage return.

34. The second form of the INSERT request allows quick insertion of a short text fragment as a line. When combined with the margin symbol, this feature also allows rapid insertion of several short lines of text.

<u>Changing Text Within a Line</u>

In order to change a piece of text in one or more lines you say:

```
MAKE (n/ALL) =text=  =new-text=
```

This request is different from all the previous requests in that it operates on text *within* lines rather than whole lines themselves. Starting from the current line, the next n (or ALL) occurrences of the text given are replaced by the new text given.

If no new text is given between the second pair of brackets, each occurrence of text will be deleted. The two bracketing symbols between the text and the new text may be compressed into a single bracket for brevity's sake.[35]

35. Again, the difficulty of reconciling notation and natural language is apparent.

<u>Editing Assumptions</u>

Just as all the editing requests depend on the assumed file for editing, there are other kinds of assumptions that affect editing. The first kind of assumption allows you to give special meanings to certain symbols when you include them in text. These "special-symbols" can make it easier for you to describe text.

<u>Referring to a Line Boundary</u>

Sometimes it is useful to refer to text at the left or right margin of a line. To do this you must first define a special symbol to represent either margin by saying:

```
ASSUME MARGIN IS special-symbol
```

If "$" is your margin symbol, then

```
=$XXX=
```

refers to a piece of text "XXX" at the beginning of a line,

=YYY$=

refers to a piece of text "YYY" at the end of a line,

=YYY$$XXX=

refers to a piece of text "YYY" at the end of a line followed by a piece of text "XXX" at the beginning of the next line, and

=ABC=

refers to a piece of text "ABC" that makes up a whole line.[36]

The special symbol consists of one to three characters. You may select any character provided that it is not a letter, digit, space, or semi-colon. You may redefine the margin symbol at any time, or you can say:

ASSUME MARGIN IS NULL

which means that *no* character will be interpreted as a MARGIN.

36. We feel that the concept of a margin symbol is important in a program editor, especially for easy reference to text at the beginning or end of a line. Nevertheless, there is a rather difficult question about text overlapping lines.

For example, should there be *two* distinct symbols for the margin characters: one for the left margin and one for the right margin; or should a special end-of-line symbol be introduced? And what about searching for text without regard to line boundaries? In the design of the Assistant there is a single symbol to denote either the left or right margin. Two margin symbols are required for text that overlaps lines, and the presence or absence of line boundaries in the pattern must be matched in the text. The matter is far from satisfactorily resolved.

<u>Using *Ellipses*</u> When referencing a long piece of text, it is tiresome to have to type it all out when only a few details identify it uniquely. In prose we use three dots as an ellipsis to indicate that a piece of text has been omitted. For example, we might quote the previous sentence as: "In prose we ... indicate that a piece of text has been omitted." With the Assistant you may omit pieces of text using a special ellipsis symbol.

For example, if you have defined "..."[37] as your ellipsis symbol, then =XXX...YYY= refers to any piece of text starting with "XXX" and ending with "YYY", and =XXX...YYY...ZZZ= refers to any piece of text starting with "XXX", ending with "ZZZ", and having "YYY" somewhere in between. You define the special ellipsis symbol by saying:

ASSUME ELLIPSIS IS special-symbol

The ellipsis may be redefined at any time, or you can say:

ASSUME ELLIPSIS IS NULL

in which case no special symbol will be defined as the ellipsis.

37. A small but important detail in the human engineering of the Assistant: The user can actually use the familiar "..." to denote an ellipsis.

<u>Referring to Character Position</u> Sometimes when referencing existing text, it is necessary to be able to refer to a character position rather

than a specific character. For example, suppose you wanted to find misspellings of the word "PASCAL." You might want to refer to something like "P_SC_L", where the underscores ("_") indicate that any single character is acceptable. You can define a special symbol which has this "wild card" meaning by saying:

```
ASSUME JOKER IS special-symbol
```
[38]

The joker may be redefined at any time, or you can say:

```
ASSUME JOKER IS NULL
```

in which case no symbol will be defined as the joker.

38. While the joker can be used in conjunction with the margin character to refer to column positions in a limited way, it is far from being a satisfactory solution. Although this is an important problem in text editing, we did not pursue it very far because its importance seemed limited for programmers.

<u>Assuming Limits</u> The second kind of assumption that affects the editing requests enables you to limit the range of all subsequent editing requests. Its form is:

```
ASSUME UPPERLIMIT/LOWERLIMIT
    IS CURRENTLINE/NULL
```

If CURRENTLINE is specified as the UPPERLIMIT or the LOWERLIMIT, then the current line becomes a boundary that all subsequent editing requests may not cross. If the UPPERLIMIT or LOWERLIMIT is specified as NULL, then that boundary is removed.

Assuming a new limit voids any previous one. Assuming a new CURRENTFILE voids all limits.

39. We believe that users learning a complex task (for example, a new computer system or a new natural language) are helped by examples. This page of the Assistant's manual gives an example of an entire user dialogue. We believe that even this example is not really sufficient for proper understanding of the Assistant's editing behavior, and that the User's Guide as a whole should probably be more example-based.

```
TEXT EDITING SESSION: The user wishes to edit an existing file called POEM.
                      (◄ indicates the current line)

--ASSUME CURRENTFILE IS POEM                              **The current line is the first line of POEM.
  What am I? ◄
--LIST ALL                                                **The entire file is displayed.
  What am I?
                                                          **This is one blank line.
  They choose me from my brothers: "That's the
  actual number of lines
  Nicest one," they said,
  Candle in my head;
  And they carved me out a face and put a
  Night was dark and will
  But when they lit the fuse, then I jumped! ◄
--PREVIOUS /actual/                                       **The current line is moved backward.
  actual number of lines ◄
--DELETE 1                                                **The current line is deleted and the line
  Nicest one," they said, ◄                               **following becomes the new current line.
--NEXT 1
  Candle in my head; ◄                                    **The current line is advanced one line.
--TRANSFER 1 INTO HOLD-FILE
  and they carved me out a face and put a ◄               **HOLD-FILE contains "Candle in my head;"
--INSERT HOLD-FILE AFTER
  Candle in my head; ◄                                    **The contents of HOLD-FILE are inserted
--ASSUME MARGIN IS $                                      **after the current line.
--ASSUME ELLIPSIS IS ...
--INSERT /$$/                                             **A blank line is inserted and becomes
    ◄                                                     **the current line.
--INSERT
++And they set me on the doorstep. Oh, the                **New lines are requested.
++                                                        **The Assistant is interrupted.
  And they set me on the doorstep. Oh, the ◄              **The last line inserted is now the
--NEXT 1                                                  **current line.
  Night was dark and will ◄
--MAKE /will/wilpqrs/                                     **One word is changed (incorrectly).
  Night was dark and wilpqrs ◄
--UNDO                                                    **The previous request is undone.
  Night was dark and will ◄
--MAKE /will/wild;/                                       **The word is changed again (correctly
  Night was dark and wild; ◄                              **this time).
--NEXT 1
  But when they lit the fuse, then I jumped! ◄
--MAKE /fuse...jumped!/candle, then I $$Smiled!$/         **The last line is altered and another line
  Smiled! ◄                                               **is added by using the MARGIN symbol.
--PREVIOUS ALL
  What am I? ◄                                            **The current line is moved back to the first
--LIST ALL                                                **line in POEM, and the entire file is listed.
  What am I?

  They choose me from my brothers: "That's the
  Nicest one," they said,
  And they carved me out a face and put a
  Candle in my head;

  And they set me on the doorstep. Oh, the
  Night was dark and wild;
  But when they lit the candle, then I
  Smiled! ◄
--PRESERVE POEM                                           **The new version of POEM is preserved.
```

Figure 1: Text Editing Session 39

File Requests

<u>Preserving Files</u> Files are normally preserved only during the dialogue at the end of your terminal session. However, if you are wary of erratic behavior on the part of the Assistant or do not feel at all confident of reaching the end of your session, then you may explicitly preserve files at any time by saying:[40]

```
PRESERVE (file-list)
```

If any of the files named in the file list do not exist or have not been changed since last preserved, then no action will be taken. You should either correct the request or enter a new request. (See "Robot's Rules of Order - Immediate Request Correction.")

40. PRESERVE also provides the user with a defense against an unreliable environment. However, if a system is subject to frequent crashes and the user must frequently interrupt his dialogue to save his work, the result will be a considerable waste of both the computer's and the user's time. Thus, reliability is also a significant human engineering concern.

<u>Restoring Files</u> If any files that have been previously preserved are changed in any undesirable way, then you always have the recourse to restore those files to their most recently preserved condition by simply saying:

```
RESTORE (file-list)
```

If any of the files in the file list have not been previously preserved or if any of them have not been changed since they were last preserved, then none of them are restored, and you should proceed as above to correct or reissue the request.

<u>Destroying Files</u> If you no longer wish to keep a preserved file or if you run out of storage space and must discard some files, then you may completely and permanently annihilate any file by saying:

```
DESTROY (file-list)
```

Beware. Once a file is destroyed, there is no way of getting it back very easily. Spare yourself some agony and make sure that you want a file destroyed before you destroy it.[41]

41. A secondary protection feature that might make this warning unnecessary would be the automatic archiving (for a time) of every file to be destroyed. This was one of the few instances in which implementation considerations were allowed to restrict the design. The archiving of destroyed files now seems to be a less formidable requirement.

<u>File Assumption</u> All of the editing requests in the previous section depend on having a "currently assumed" file to edit. In order to specify what file is to be assumed simply say:

```
ASSUME CURRENT FILE IS file
```

Except where noted, all other requests use the assumed file if a file is not given explicitly.

<u>File Renaming</u>

In order to change the name of the CURRENTFILE, all you have to say is:

```
ASSUME NEWNAME IS new-file-name
```

Program Requests

Executing Programs

In order to execute a PASCAL program say:[42]

```
RUN (file-list) (WITH parameter-
                      file-list)
```

The "parameter-file-list" is a list of file names that are to be substituted for the formal file parameters in the program header of your PASCAL program. If a file exists in your program header but is omitted from your parameter-file-list, then the file name assumed is that of the formal parameter in the program header. For example, if your program header is:

```
PROGRAM DUMMY(FILE1,FILE2,FILE3,
              FILE4,FILE5);
```

and you type:

```
RUN DUMMY WITH XYZ,,ABC,DEF
```

then your request will be interpreted as:

```
RUN DUMMY WITH XYZ,FILE2,ABC,DEF,
               FILE5
```

If more than one file name is given in the file list, the first file named is assumed to contain the main program segment, and all the others to contain external procedures. For further information on the linking of externals to PASCAL programs, see "Appendix 2: Linking External Procedures to PASCAL Programs."

If a PASCAL error exists in your program, you will be told so, and your program will not be executed. To see a listing of those errors use the VERIFY request described below.

42. The spirit of the RUN request is that it runs a PASCAL program. The form that the program is in is irrelevant. If need be, the program will be compiled, but this is transparent to the user unless errors are found. The mechanics of keeping track of source and object versions if they are distinct is managed automatically by the Assistant. Of course, complete control of the computer passes to the user's program and the Assistant disappears. From our point of view, this is bad; but the alternative, incorporating a kernel Assistant into the run-time program, seemed overwhelming Building a kernel of the Assistant into the PASCAL run-time system now seems inescapable, despite the implementation difficulties.

Verifying Programs

In order to get a summary of errors in your PASCAL program just say:

```
VERIFY (file) (INTO file)
```

Depending on whether you have assumed a TERSE or LONG mode of interaction, you will get either a brief summary of error messages, a more detailed summary of errors, or a full listing of your program with error messages. If "INTO file" is specified, the verification will be put into the file instead of being displayed at the terminal.[43]

43. Complementary to RUN, the VERIFY request is strictly for checking a program. Object code might be generated but that is the Assistant's business, not the user's.

Formatting Programs

In order to format your PASCAL program according to standard pretty-printing conventions say:

```
FORMAT (file) (INTO file)
```

If "INTO file" is specified, the results of the format will be put into that file; otherwise, they will be displayed at the terminal.

The FORMAT request takes a text file containing a PASCAL source program and reformats it according to a set of standard spacing conventions. FORMAT in no way affects the logical ordering of the program; it merely rearranges the file into a standard format. The standards have been developed so that the reformatted program is aesthetically appealing, logically structured, and above all, readable.

Extra spaces and extra blank lines found in the text are kept. You may improve the readability of your program even more by adding extra spaces and blank lines beyond those inserted by the Assistant.[44]

44. The FORMAT request is based on a program that automatically prettyprints PASCAL text. A detailed description of this program appears in Hueras (1976). This program contains several features that we believe are unique. For one, the program needs no information from the user other than the file itself. For another, the program handles even program fragments. Our initial feeling was that developing an automatic formatting program was easy. This did not turn out to be the case.

For example, if your currentfile looks as follows:

```
TYPE SCALE = (CENTIGRADE, FAHRENHEIT);

FUNCTION CONVERT( (* FROM *) DEGREES: INTEGER;
(* TO *) NEWSCALE: SCALE): INTEGER;
BEGIN IF (NEWSCALE = CENTIGRADE) THEN
CONVERT:=ROUND((9/5*DEGREES) + 32) ELSE
CONVERT:=ROUND(5/9*(DEGREES - 32)) END;
```

and you then type "FORMAT", the reformatted program will be printed at your terminal as follows:

```
TYPE SCALE = (CENTIGRADE, FAHRENHEIT);

FUNCTION CONVERT( (* FROM *) DEGREES: INTEGER;
                  (* TO *) NEWSCALE: SCALE): INTEGER;
BEGIN
   IF (NEWSCALE = CENTIGRADE)
      THEN
         CONVERT := ROUND((9/5*DEGREES) + 32)
      ELSE
         CONVERT := ROUND(5/9*(DEGREES - 32))
END;
```

<u>Binding Programs</u>

There may come a time when you simply won't be modifying a program any further, but executing it very often. For execution efficiency you may bind your program into an execute-only file by saying:

BIND (file-list) INTO file[45]

45. A vestigial concession to manual program management strategies. On a high level architecture like the Burroughs B7500 it would be irrelevant. (We may have been shortsighted as it now seems to us that even a conventional loader environment could probably be effectively managed automatically by the Assistant.)

If there are any PASCAL errors in any of your programs, the programs will not be bound and you will be informed of your situation.

Robot's Rules of Order

Prompting Messages

1. Two prompting characters are always printed by the Assistant to indicate its attentiveness. The characters indicate what type of response is expected from you.

Prompting Characters	*Response Type*
(Note: "ƀ" signifies a space)	
--[46]	Requests
++	New-Text Input
//	Caution Checks
?ƀ	PASCAL Program Input

46. Another detail. We spent a lot of time trying to choose meaningful and distinctive graphics for these prompting symbols because they will be seen so frequently.

Information Requests

2. An information request may be issued whenever the Assistant is attentive, regardless of what prompting message has been given. The only exception is the "?ƀ" prompt, which is issued by a PASCAL program, *not* the Assistant.[47]

47. Probably our darkest hour.

File Names

3. File names may be of arbitrary length, but no less than two characters. The characters that may be used are letters, digits, and the blank character "-". The first character must be a letter and the last cannot be the break character.[48]

For example:

```
SQUARE-ROOT-PROGRAM
SINE-COSINE-FUNCTION
```

are legitimate file names, while names such as:

```
3QX  (does not begin with a letter)
A    (contains too few characters)
H3.I (contains an illegal character)
```

are not.

48. The break character for compound names in natural language is the hyphen. Thus, for the request language we use the hyphen to connect compound names. We believe this convention is easy to use and well-founded.

Abbreviations and Request Spacing

4. All words in requests, with the exception of file names and the request names PRESERVE, RESTORE, and DESTROY may be abbreviated by their first letter.[49] Spaces in a request may be omitted, with the exception that files and file lists must be preceded and followed by a space.[50]

For example:

```
TRANSFER 3 INTO ALPHA
NEXT 5
```

49. There are a number of two-word keywords, like CURRENT-FILE, in the request language. It is certainly not clear how to abbreviate them.

50. The requirement that spaces delimit file-names was intended to eliminate ambiguity. Ambiguity now seems rare enough to be worth tolerating.

may be abbreviated as:

```
T31 ALPHA
N5
```

<u>Multiple Requests</u> 5. You may type in more than one request on a line any time by separating each request by a semicolon (";").[51]

<u>Interaction Control</u> 6. Each of the words TERSE, LONG, CAUTIOUS, or RISKY may be appended to any request on a line to temporarily override the currently assumed mode of interaction for the duration of the request.[52] For example, if you are currently assuming LONG messages but would rather not see a LONG message for an EXPLAIN request, then you would type:

```
EXPLAIN (name) TERSE
```

<u>Immediate Request Correction</u> 7. Whenever a request is given and not satisfied due to an error, you may correct the error by modifying the request, rather than retyping it entirely. To do so simply type:

```
=old-text=new-text=
```

In this case, "new-text" will replace the first occurrence of "old-text" found in the erroneous request, and the Assistant will then automatically attempt to satisfy the request again for you. If old-text is not found in the erroneous request, then nothing is done, but you still have the option of trying to modify the request once more. "=" may be replaced by any character other than a letter, or ";", which is neither in old-text or new-text. It is used simply as a separator and is not considered part of either old-text or new-text.

51. Allowing multiple requests on a line enables the more experienced user to build compound requests. In an environment with slow reaction time it may give the user more satisfaction to work with longer request lines and adapt to the slower pace. As Palme (1975) and others have pointed out, such adaptation is comparable to the adaptation that takes place in natural human dialogue.
This feature is not novel, but the Assistant's interactive "chatter" during execution of a request line and the immediate request correction facility make it more effective.

52. There is some doubt in our minds as to the value of this.

Appendix 1: The Assistant at a Glance

General Requests:

```
EXPLAIN     (name)

SHOW        (name)

ASSUME      INTERACTION   IS   TERSE / LONG
            SECURITY      IS   CAUTIOUS / RISKY
            CURRENTFILE   IS   file
            NEWNAME       IS   new-file-name
            MARGIN        IS   special-symbol / NULL
            ELLIPSIS      IS   special-symbol / NULL
            JOKER         IS   special-symbol / NULL
            UPPERLIMIT    IS   CURRENTLINE    / NULL
            LOWERLIMIT    IS   CURRENTLINE    / NULL
GRIPE

UNDO

QUIT        (QUICKLY)
```

Editing Requests:

```
NEXT        (n/ALL)  (=text=)

PREVIOUS    (n/ALL   (=text=)

LIST        (n/ALL)  (=text=)

DELETE      (n/ALL)  (=text=)

TRANSFER    (n/ALL)  (=text=)   INTO file

COPY        (n/ALL)  (=text=)   INTO file

INSERT      (=text= / file)     (BEFORE / AFTER / OVER)

MAKE        (n/ALL)   =text=    =new-text=
```

File Requests:

```
PRESERVE    (file-list)

RESTORE     (file-list)

DESTROY     (file-list)
```

Program Requests:

```
RUN         (file-list)         (WITH parameter-file-list)

VERIFY      (file)              (INTO file)

FORMAT      (file)              (INTO file)

BIND        (file-list)          INTO file
```

Request Modifiers:

```
TERSE / LONG
CUATIOUS / RISKY
```

Request Correction:

```
=old-text=new-text=
```

Request Spacing:

Requests	- - Spaces in a request may be omitted, with the exception that files and file-lists must be preceded and followed by a space.
File-names	- A file-name must be comprised of at least two characters. Characters that may be used are letters, digits, and the break character ("-"). The first character of a file-name must be a letter, and the last character cannot be a break character.
File-lists	- A file-list is a list of file-names separated by commas (",").
Multiple Requests	- More than one request may be typed on a line provided that each request is separated by a semicolon (";").

Prompting Characters and Response Types:

--	Requests
++	New Text Input
//	Caution Checks
?ƀ	PASCAL Program Input

Conventions

1. Uppercase letters denote reservedkkeywords.
2. Lowercase letters denote objects.
3. Parentheses denote optional keywords or objects.
4. A slash ("/") denotes mutually exclusive alternatives.
5. "ƀ" denotes a space.
6. All keywords may be abbreviated by their first letter, except for PRESERVE, RESTORE, and DESTROY.

Acknowledgments

We would like to express our appreciation to Dr. Conrad Wogrin and the University Computing Center for their generous support and interest in this project.

Many students have also contributed to this work. Daryl Winters, Randy Chow, Peter Haynes, Gary Madelung, and Dave Tarabar helped design and build early prototypes of major system components. The students of COINS 790T and 7900 have all contributed to the ideas in this work.

Finally, our thanks to Michael Marcotty, who has continually stimulated our efforts to develop a pleasant environment for users.

We are grateful to the National Science Foundation and the U.S. Army Research Office for their support of this effort.

References

[Boies 1974]

Steven J. Boies
"User Behavior on an Interactive System,"
IBM Systems Journal, No. 1, 1974.

[Chen-Chance 197-]

Selvin Chen-Chance
Doctoral Dissertation, University of Hawaii, Department of Educational Psychology

[Child et al. 1961]

Julia Child, Louisette Bertholle, and Simone Beck
MASTERING THE ART OF FRENCH COOKING
Vol. 1, Alfred A. Knopf, New York, 1961.

[Cooke and Bunt 1975]

John E. Cooke, and Richard B. Bunt
"Human Error in Programming: The Need to Study the Individual Programmer,"
Department of Computational Science Technical Report 75-3, University of Saskatchewan, Canada, 1975.

[Cuandra 1971]

Carlow A. Cuandra
"On-Line Systems: Promise and Pitfalls,"
JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE, March-April, 1971.

[Freedman and Landauer 1966]

J. L. Freedman, and T. K. Landauer
"Retrieval of Long-term Memory: Tip-of-the Tongue Phenomenon,"
PSYCHOLOGICAL SCIENCE, Vol. 4, No. 8, pp. 309-310, 1966.

[Gilb and Weinberg 1977]

Thomas Gilb, and Gerald Weinberg
HUMANIZED INPUT
Winthrop Publishers, Cambridge, Mass., 1977.

[Hoare 1976]

C.A.R. Hoare
Private communication, 1976.

[Holt and Stevenson 1977]

H. O. Holt, and F. L. Stevenson
"Human Performance Considerations in Complex Systems,"
SCIENCE, Vol. 195, pp. 1205-1209, 1977.

[Hueras and Ledgard 1977]

Jon Hueras, and Henry Ledgard
"An Automatic Formatting Program for Pascal,"
SIGPLAN NOTICES, Vol. 12(7), pp. 82-84, July 1977.

[Kennedy 1974]

T.C.S. Kennedy
"The Design of Interactive Procedures for Man-Machine Communication,"
INTERNATIONAL JOURNAL OF MAN-MACHINE STUDIES, Vol. 5, pp. 309-334, 1974.

[Kid and Van Cott 1972]

Jerry S. Kid, and Harold P. Van Cott
"System and Human Engineering Analyses,"
Chapter 1 in HUMAN ENGINEERING GUIDE TO EQUIPMENT DESIGN, H. P. Van Cott and Robert Kinkade, editors, US GPO, Doc. D4.10:EN3, Washington, D.C., 1972.

[Ledgard and Singer 1977]

Henry Ledgard, and Andrew Singer
"Formal Definition and Design,"
in preparation.

[Mann 1975]

William C. Mann
"Why Things Are So Bad for the Computer-Naive User,"
PROCEEDINGS OF THE NATIONAL COMPUTER CONFERENCE, pp. 785-787, 1975.

[Miller and Thomas 1977]

Lance Miller, and John C. Thomas, Jr.
"Behavioral Issues in the Use of Interactive Systems: Part I. General System Issues,"
Thomas J. Watson Research Center, Yorktown Heights, N.Y., 1977.

[Palme 1976]

Jacob Palme
"Interactive Software for Humans,"
Research Institute of National Defense, Stockholm, Sweden, NTIS NO.
PB-245 553, July 1976.

[Parsons 1970]

Henry M. Parsons
"The Scope of Human Factors in Computer-Based Data Processing Systems,"
HUMAN FACTORS, Vol. 12, No. 2, pp. 165-175, 1970.

[Singer and Ledgard 1977]

Andrew Singer, and Henry Ledgard
"The Case for Human Engineering,"
Computer and Information Science Department Technical Report 77-11,
University of Massachusetts, Amherst, Mass., September 1977.

[Stemple 1975]

David Stemple
Private communication, 1975.

[Sterling 1974]

Theodor D. Sterling
"Guidelines for Humanizing Computerized Information Systems: A Report
From Stanley House,"
COMMUNICATIONS OF THE ACM, Vol. 17, No. 11, November 1974.

[Teitelman 1974]

Warren Teitelman
"Interlisp Reference Manual,"
Xerox Corporation, Palo Alto Research Center, Palo Alto, Calif, 1974.

[Thorndike and Rock 1934]

Edward L. Thorndike, and Robert T. Rock, Jr.
"Learning Without Awareness of What is Being Learned or Intent to Learn It,"
JOURNAL OF EXPERIMENTAL PSYCHOLOGY, Vol. XVII, No. 1, 1934.

[Turner 1974]

Rollins Turner
"Interaction Data From CS/2,"
Digital Equipment Corporation, Maynard, Mass., 1974.

[Vandenberg 1967]

J. D. Vandenberg
"Improved Operating Procedures Manuals,"
ERGONOMICS, Vol. 10, No. 2, pp. 114-120, 1967.

[Weinberg 1971]

Gerald Weinberg
THE PSYCHOLOGY OF COMPUTER PROGRAMMING
Van Nostrant Reinhold Company, 1971.

[Wiedmann 1974]

Clark Wiedmann
HANDBOOK OF APL PROGRAMMING
Petrocelli Books, New York, 1974.