AIX usability enhancements and human factors

by F. C. H. Waters
R. G. Bias
P. L. Smith-Kerker

As microcomputers become capable of running increasingly large and complex operating systems, the question of the usability of those operating systems becomes critical. Most microcomputer users neither are nor want to be systems programmers, yet most of the existing large operating systems assume the existence of a dedicated systems programming organization to install and maintain system software. This paper describes the process by which a large existing operating system was modified to allow it to be installed, configured, maintained, and used by individuals with minimal programming knowledge. We describe the aspects that had to be changed, the kinds of modifications that were required, the reasoning behind those modifications, and the priorities that constrained our activity. We also describe the development process by which potential usability problems were identified and corrections were defined, implemented, and validated.

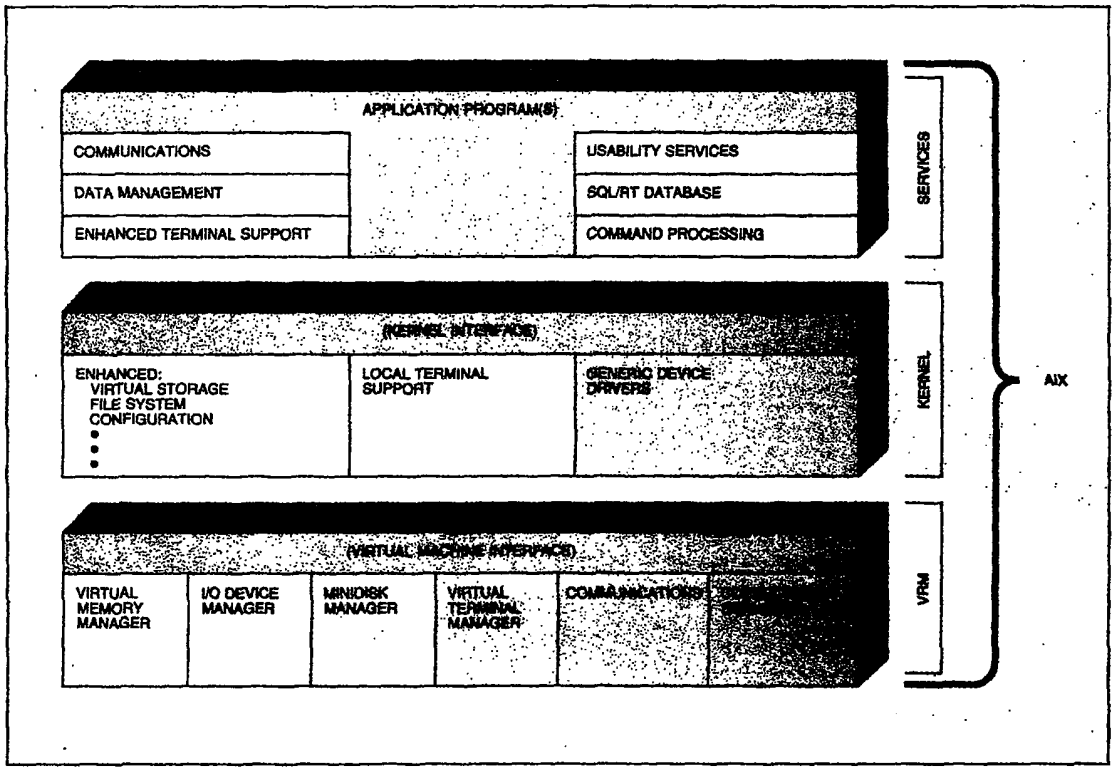
The RISC Technology Personal Computer (RT Personal Computer)™ is relatively new to the IBM product line, having been announced in January 1986. (RT Personal Computer is a trademark of International Business Machines Corporation.) The RT Personal Computer combines a processor architecture that is known as Reduced Instruction Set Computer (RISC) with an operating system known as the Advanced Interactive Executive (AIX)™.[1] (AIX is a trademark of International Business Machines Corporation.) The core of AIX is a modified version of the UNIX®[2] System V operating system, selected to allow a number of strategic applications to be migrated to the RT Personal Computer with minimal reprogramming. (UNIX is developed and licensed by AT&T, and is a registered trademark of AT&T in the U.S.A. and other countries.) AIX is not simply a UNIX variant, however. The designers disciplined themselves to maintain compatibility with the UNIX Application Program Interface (API) and to provide high quality, usability, and performance. The designers made substantial changes within the UNIX component, added function above the API level, and provided a virtual resource manager under the UNIX kernel to provide a coherent virtual machine interface that had no logical home within the kernel proper. Figure 1 shows the overall structure of AIX, in which the section labeled "services" contains most of the components that interact directly with the user. The service labeled "command processing" is the UNIX shell.

The human factors challenge

Any designer of a human–computer interface encounters a conflict between the objectives of the novice and the expert user. A novice needs clear direction, tutorial guides to operations, logical consistency in the human–computer interface, and a minimal short-term individual memory requirement. On the other hand, an expert is impatient with mechanical requirements that increase the number of keystrokes or processing delays. Also, experience teaches that today's novice often becomes tomorrow's expert. One of our main objectives has been to design for a smooth transition from novice to expert.

In the case of AIX, the fact that we could change neither the command language itself nor the applications that would be ported to AIX from other UNIX-

Figure 1 Overall structure of the AIX operating system



like operating systems caused us to concentrate in other areas. We concentrated our design and implementation resources on those tasks that very few users do often enough to become experts at—installation, configuration, and system management—which we considered the weakest points of the UNIX human–computer interface. Thus we could indulge in highly tutorial interfaces without too much concern for the impatience of experts.

The choice of a UNIX base was particularly challenging from a human-factors point of view. The UNIX operating system was originally designed as a powerful and flexible tool for computer science experimentation.[3] To users for whom a computer is a means and not an end, however, the UNIX command language can seem complex and unpredictable. The very open-endedness that has allowed continuing expansion of the UNIX functional power over the years has resulted in a wide variety of syntaxes and semantic characteristics. The need to make a command useful both in *Shell scripts* (i.e., the UNIX equivalent of VM EXECs) and when issued from the terminal has resulted in responses that often seem erratic to unsophisticated terminal users. For example, the default output device for the *pr* (print files) command is not a printer but Standard Out (i.e., the display or the input end of a pipe). This is due to the fact that the *pr* command is really a formatting, not a printing, command. Thus, a user who wants a file printed must pipe the output of the *pr* command into a command that actually prints it. For example, the command *print* enqueues files on the printer queue. However, if the user says *print* only, the file is printed but without formatting or pagination.

The UNIX user interface was originally designed for typewriter-like terminals connected to a minicomputer via low-speed lines. Many of the characteristics that limit the UNIX usability today—such as extreme

terseness of command language—resulted from design decisions intended to improve the performance of the early UNIX systems.

The process of installing and customizing a UNIX system is also quite complex, requiring an understanding of UNIX internal structures and processing. Adding a new device to the configuration, for example, requires the user to edit a number of configuration files to define the new device. This configuration interface is another vestige of the use of the UNIX system primarily by system programmers. Early users of the UNIX system needed and were competent to use complete access to all of its internal files. We believed it to be essential to simplify the process of installation and configuration for the majority of our users, who might or might not be professionally capable of installing a UNIX system, but who in any case had more urgent objectives than learning the UNIX structures merely to install a printer. At the same time, we wanted to preserve the UNIX openness for the user who really needs its flexibility. Our guiding principle was that standard things should be easy; complex things should be possible.

We believed it to be essential to simplify the process of installation and configuration for the majority of our users.

Our user interface design was subject to a number of constraints, some architectural and some practical. The following are some of the guidelines we observed:

- Those who use the RT Personal Computer primarily to run one specific application need an interface that enables them to install and configure the system and application, manage files, and perform routine system functions without requiring the complexities of the full command language.
- Except for the installation interface, all of the system user interfaces are to be available in substantially the same form on all of the RT Personal Computer console displays (including the PC monochrome, character-oriented display) and on attached Asynchronous terminals. Applications operate only on those terminals and/or displays capable of satisfying the functional requirements of the particular application.
- The user must be able to switch into and exploit the full command language when necessary.
- The ability of the RT Personal Computer system to run multiple, concurrent, interactive sessions is an inherent part of the user interface.

The design process

One of the earliest and most significant decisions in the design of the usability improvements to AIX was the inclusion of human factors professionals as members of the design team. Two of the authors (Bias and Smith-Kerker) took offices in the development area and participated in the day-to-day design decisions. This placing of human factors professionals in the development mainstream is in contrast to the more common approach of having a human factors group act as detached observers and consultants.

Having human factors professionals organizationally part of a separate department, but working and having offices among the developers guards against three potential (and sometimes actual) problems. This way, human factors people are likely to be perceived as committed team members by the developers. Thus, the day-to-day human factors work is done among the developers and is visible to them. Also, these professionals are not likely to become uncritically sympathetic to the development constraints because they continue to interact with their human-factors peers and management. Third, the immediacy of interaction allows the detection and correction of potential usability problems in the first release.

Early in the development of the RT Personal Computer, most of the human factors effort was directed at providing input to the department responsible for the development of the user interface. This group, of which one author (Waters) was a member, was responsible for developing a user interface specification. The specification consisted of detailed information on how the user interface was to function, including presentation of displayed information, methods by which users interacted with the system, error message content, and help-panel content. Everyone developing user interfaces for the RT Per-

sonal Computer was to adhere to this user interface specification.

One of the earliest human factors efforts for the RT Personal Computer was the development of a simulation model of the user interface, as described by the user interface specification. This model proved to be an excellent tool for validating the user interface specification and revealing and resolving mismatches between the specification and actual experience.

One of the earliest human factors efforts for the RT Personal Computer was the development of a simulation model of the user interface.

Although the user interface specification was quite detailed, most people still had difficulty developing an accurate mental picture of what the user interface would look like and how it would function. The model allowed people in the development organization to actually see the user interface and use it to a limited extent. The model also made it much easier to understand the complex interactions among the user interface elements. Finally, the user interface model allowed us to discover problems with the interface that were not obvious from reading the specification.

In addition to employing the user interface model to review and augment the written specification, the model was used to make design decisions. In some cases, possible interface designs were programmed, and the resulting interfaces were informally evaluated (in the form of an *expert review*) in order to select the best design. In an expert review, possible designs were shown to human-factors and user-interface experts to help eliminate obvious flaws and thus narrow the scope of naive-user testing. However, it was not always clear which alternative was the most usable. In these cases, reviews of the design alternatives were supplemented by user testing.

User testing involved having test subjects execute tasks on the different versions of the model. After reading some instructional material and practicing briefly, each subject completed specific tasks that used the elements being evaluated. As the subjects completed the tasks, performance data were collected in the form of length of time to complete each task and number and type of user errors. Subjects also completed questionnaires soliciting their subjective evaluations of the user interface.

Some of the model-based tests were preceded by paper-and-pencil tests to allow us to narrow the variations to be modeled. The main result of these tests was to make us much more wary of paper-and-pencil methods in general, because their results were frequently at odds with those of the subsequent model-based user tests.

The main design alternative studies were the following six: (1) which symbols to use for the various functions; (2) different methods for selecting user interface elements from the screen; (3) various input devices for the selection of user interface elements; (4) the performance of users with several different mouse input devices; (5) subjective preference for the size and shape of mouse input devices; and (6) symbols to use for indicating a user's relative location in a document while scrolling.

As the project proceeded into the detailed specification stage, a software usability work group was created. This work group included permanent members from the following other groups: user interface design, development human factors, systems assurance human factors, and information development and design. Software developers attended those meetings that concerned their own components. The members of the work group developed and walked through a series of scenarios of critical points in the use of the operating system.

The clearest accomplishment of the software usability work group was that many problems were identified, and the solutions recommended by the group were accepted and implemented. However, there were also more subtle outcomes from the work of this group. Primarily, the group provided an excellent basis for developing good working relationships among the people responsible for usability and the individual software developers. Specifications of the software had not been completed during the time in which the scenarios were being documented. Consequently, members of the group worked with indi-

vidual developers a great deal in order to understand and document details of the software user interface. This interaction provided an excellent arena for establishing a cooperative, helpful relationship. The developers came to understand that the purpose of the software usability work group was to help produce a better product, not to find fault with individual efforts. They also became aware that the comments from the group were based on systematic analysis and were not merely differences of taste.

As the system implementation proceeded, the human factors professionals on the project turned much of their attention to system usability testing. They adopted what has been referred to as a find-and-fix methodology. That is, while the product did have measurable usability criteria against which it would eventually be tested (e.g., number of minutes to install the operating system), the human factors concern at this stage was less with criteria and descriptive statistics than with particular errors made by test subjects.

System usability testing involved placing representative users (as specified in product planning audience descriptions) in representative environments and asking them to perform representative tasks. The human factors professionals did collect times to complete and error rates on all the tasks, as well as satisfaction data from questionnaires. However, the data of importance were the particular errors made and the groupings of these errors. That is, if a certain interface screen was giving users trouble, human factors and development team members devised an improved presentation. An important aspect of this testing was the immediate reporting of the results to the responsible developers. Because the human factors professionals were part of the team, they did not have to wait until the end of the project to make their study and write a report, and then try to persuade the developers to implement their findings in some later system release. Instead, problems and recommendations for correction were reported to developers as soon as they were identified. This minimized wasted developer effort, ensured developer participation in the design of the solution, and maximized the number of identified problems that were corrected in the first release.

Toward the end of the development cycle, the human factors professionals conducted another iterative series of tests. This series tested the customer installation process, from unboxing and cabling the hardware to installing and configuring the software. Again, the users and their environments and tasks were representative of those projected for the product in the field. Problems were reported, and fixes were designed immediately.

Thus, the human factors professionals participated intimately in the design, implementation, and final shipment of the operating system. Their real-time involvement improved the quality of many decisions that would have been almost impossible to take back later. Human factors involvement helped to avoid the frustration of unimplementable changes based on after-the-fact assessments. At the same time, their continuity in a human factors organization of their professional peers kept these professionals from being too ready to compromise in the name of expediency.

We chose to build installation dialogs that perform the mechanics of installation via programming.

Resulting user interfaces

In the process of transforming the UNIX operating system into AIX, usability enhancements were made in the following three main areas: installation, configuration, and activity management. Whereas installation and configuration may be self-explanatory, *activity management* is defined as the method by which the AIX ability to run multiple concurrent interactive sessions is controlled and used.

Installation. The installation of most UNIX systems is rather complex, requiring the user to have an understanding of the internal structure and logic of the operating system. If we had perpetuated this approach, the installation of AIX would have been even more complex because of the presence of the Virtual Resource Manager (VRM). We chose instead to build installation dialogs that perform all of the mechanics of installation via programming.

Figure 2 A typical installation panel

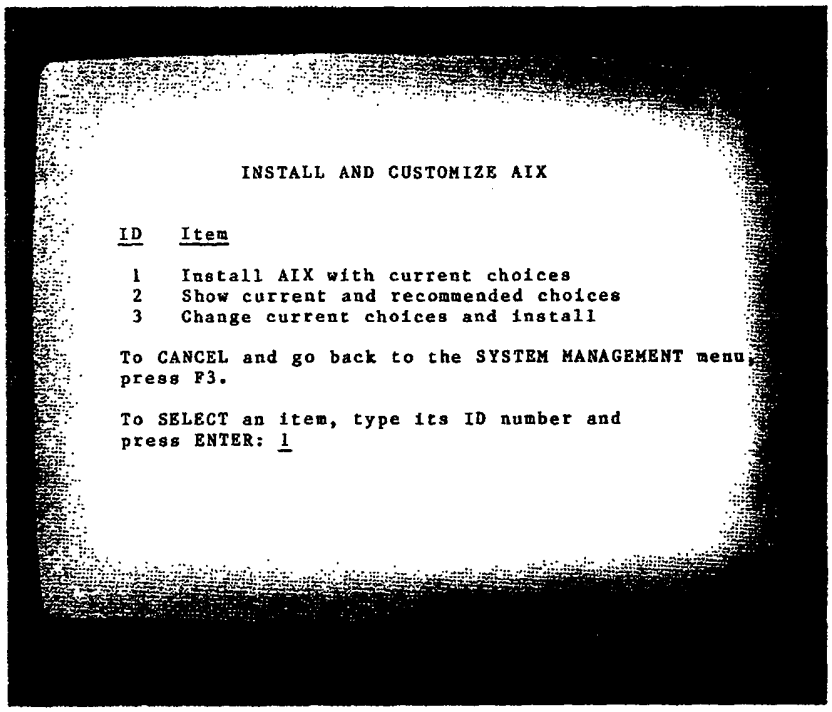
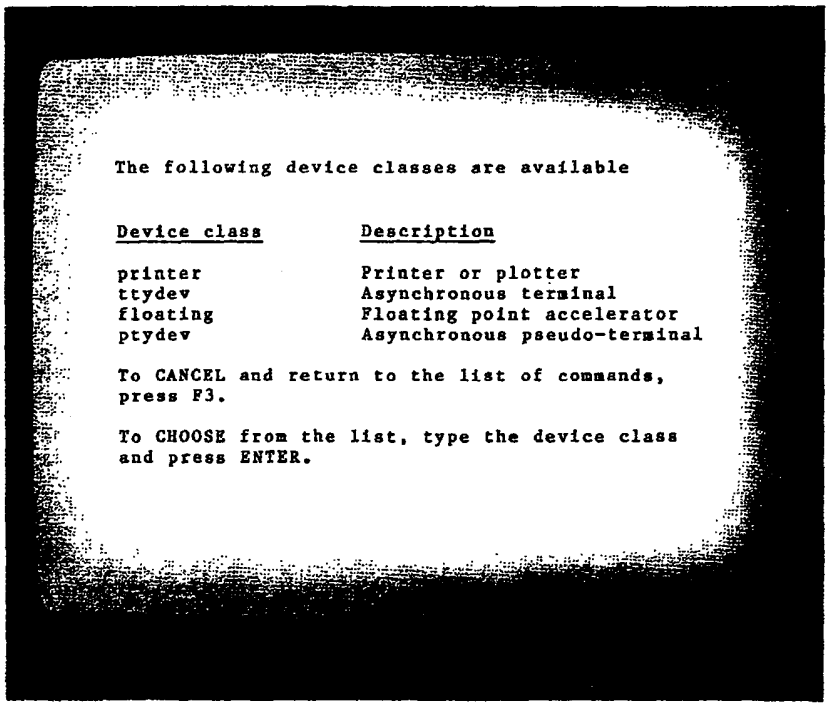


Figure 3 A typical DEVICES command screen



AIX is installed in two main stages: Virtual Resource Manager (VRM) installation and Base (UNIX) installation. This separation is required by the fact that the VRM is available as a separate, licensed program product. The two installation dialogs were implemented by two different development organizations (one within and one outside IBM), and the specific focal point for user interface consistency—the software usability work group—prevented them from diverging. The installation dialogs present the user with a series of prompts for making choices and mounting diskettes containing the operating system. A typical screen is shown in Figure 2.

Configuration. Configuring a UNIX system consists of defining the users of the system and the I/O devices that are available to them. In addition, the Virtual Resource Manager component of AIX implements the concept of virtual minidisks. Therefore, the AIX user must also be able to define and manipulate minidisks. To simplify these tasks, we defined USERS, DEVICES, and MINIDISKS commands. These commands initiate interactive dialogs with the user, determining the specific operation the user wants to perform, soliciting parameters for that operation, obtaining confirmation that the parameters have been received correctly, and then performing all of the necessary UNIX or VRM changes to ensure that the user's request is done correctly. A typical DEVICES command screen is shown in Figure 3.

The Usability Package. Since AIX is capable of running a number of independent processes simultaneously, we had to give the user a means of managing those processes. We also had to give the comparative novice a means of managing files and performing utility operations without resorting to the UNIX Shell. We achieved these objectives by implementing *virtual terminals* in the VRM and by adding to the UNIX component a *Usability Package* containing an Activity Manager and Tools and Files programs.

Each virtual terminal runs an independent process, with a separate address space and UNIX environment. A virtual terminal running a program constitutes a window onto the output of that program. Our user interface currently provides six kinds of windows, each running its own specialized user interface program. The six types of windows are the following:

- The Windows window, shown in Figure 4, is the operator's control panel. It is the first thing dis-

Figure 4 A Windows window after three other windows have been created

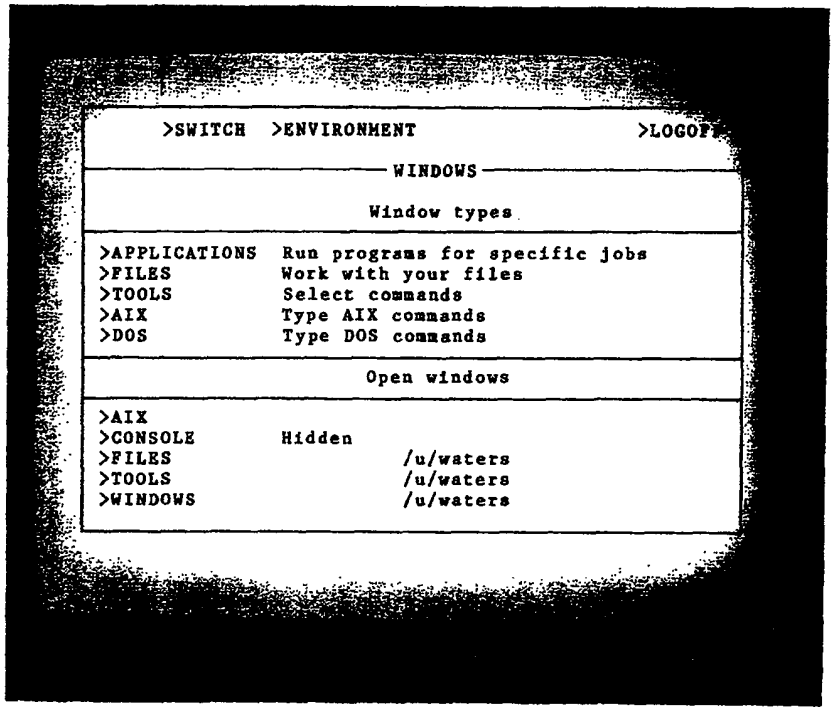
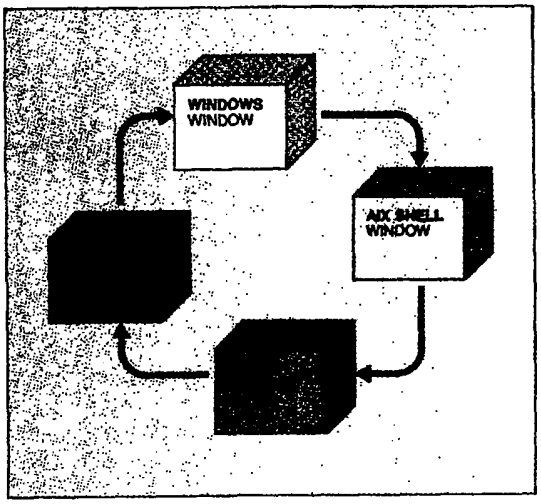


Figure 5 A ring of windows



played when the user starts the activity manager (either explicitly or implicitly at log-on), and it is the base from which all new windows are created. The top portion of the Windows window contains a list of the kinds of windows that can be created, and the bottom portion displays a list of the windows that already exist. This particular Windows window shows a situation in which the user had already created one AIX Shell window, one Files window, and one Tools window.

- The Console window is the target for messages such as system errors that are not specific to any given virtual terminal. The Console window is normally hidden and is not displayed when the user successively displays members of a ring of windows, an example of which is shown in Figure 5.
- A Files window is a full-screen display of a directory in the user's file system. Selecting a file causes the user to be presented with the set of actions that can validly be performed on that file. The Files program was added to implement an Object-Action user interface (i.e., choose an object, then specify the action to be applied to it). This reduces the user's need to remember the exact names and types of files and directories, and the commands that apply to them. An example Object-Action screen is shown in Figure 6.
- A Tools window is a hierarchically arranged list of commands and application programs that can be invoked via a panel rather than a command-language interface. The Tools program implements an Action-Object user interface [i.e., choose an action, then identify the object(s) to which it is to be applied]. This reduces the user's need to remember the exact syntax (and occasional semantic peculiarities) of commands. It also provides a panel interface to commands that deal with entities that are only metaphorically objects, such as free space in the file system. A typical Action-Object screen is shown in Figure 7.
- An AIX Shell window is the equivalent of a single instance of the AIX Shell running on an Asynchronous terminal.
- A DOS Shell window is identical to an AIX Shell window, except that it has been preconditioned to submit commands to the PC DOS compatibility interface of AIX.

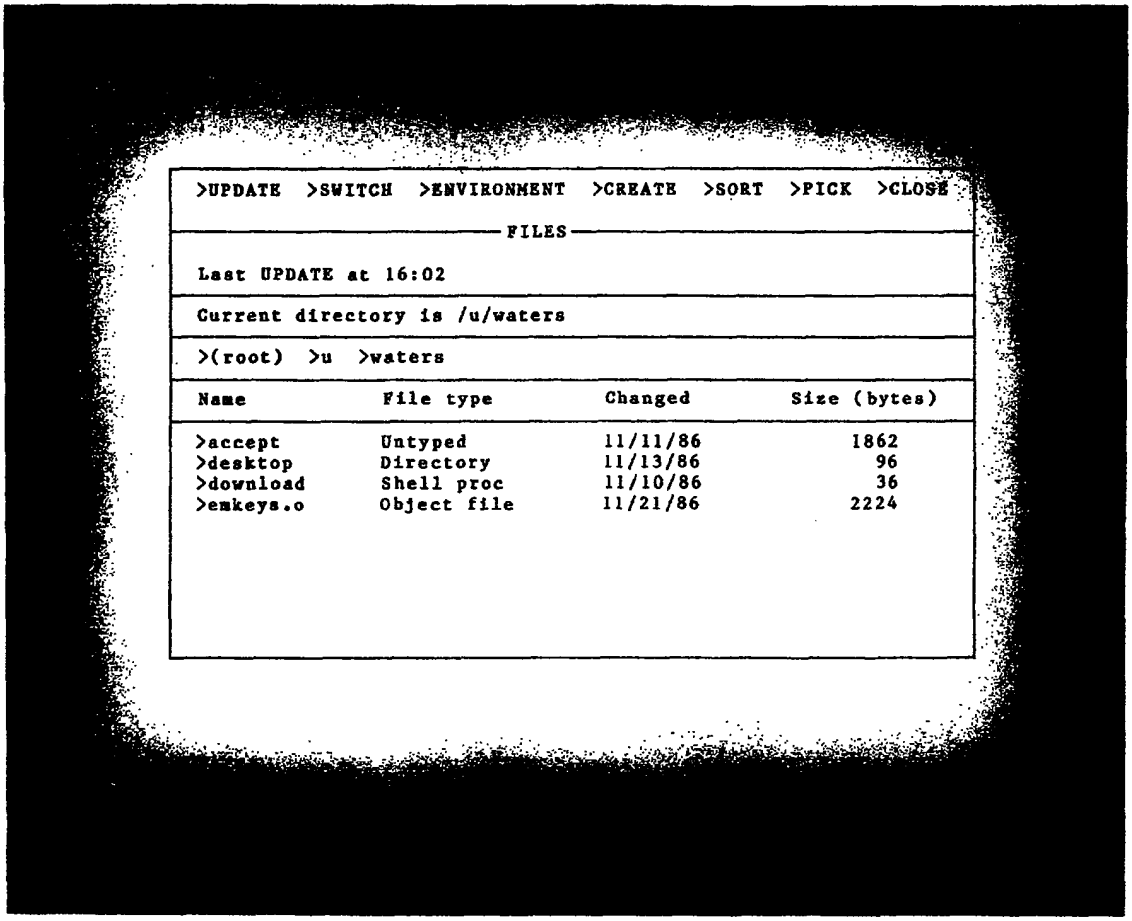
After log-on, the user can determine which interfaces are most appropriate to the tasks to be performed and then create several windows of suitable types. The windows form a ring.

The user can move around the ring of windows with the key combinations Alt-Action (forward) and Shift-Action (backward). A user who has created a large number of windows (up to the maximum of 28) can go directly to the Windows window with the Ctrl-Action key combination. The user then moves directly to the desired window by selecting it in the Windows window and selecting the ACTIVATE command, which appears on the Command Bar (the top line of the screen) when a window is selected.

Files windows. The Files program presents the user with a list of the files in the current directory. Although there are options to limit the set of files presented, to sort the list, or to select other segments of the directory tree for display, the primary operator action is to select the file to be acted on. When a selection has been made, the operator is presented with a choice of actions that apply to the chosen file. The determination of what actions are valid for a particular file is based on its file type and is controlled by a file-type description that resides in a shared data area outside the Files program.

For each file type there may be a special print program, compiler, editor, interpreter, etc. For any of these entries, the specification either may be empty, indicating that the option is not valid for that file type, or may contain the name of the program that provides the support for the function. For example, for most files the editor specified is *e*, whereas for object programs no editor is specified.

Figure 6 Files program uses Object-Action

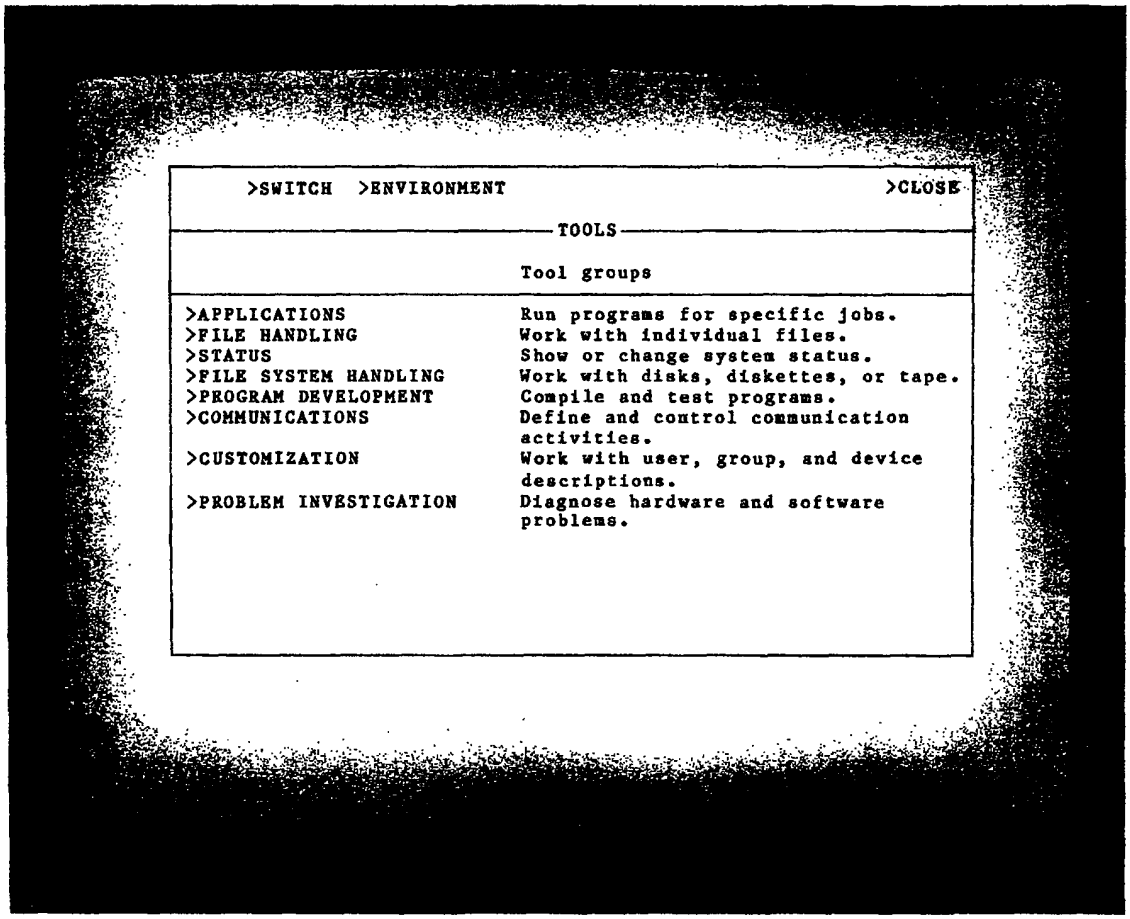


The description of a file type is carried outside the Files program. This provides a mechanism to modify file types without modification of the Files program itself. New file types can be added to the system simply by adding a description for the new file type. (An interactive program makes this addition easy.) The main Files program does not require modification unless new classes of actions are added (in addition to print, edit, compile, etc.).

Tools windows. The Tools program presents the operator with lists of actions that can be invoked. Lists of actions available are grouped into sets of related actions. The first list presented is the list of available groups. After the selection of a group, the commands/actions that are part of that group are presented. Selection of a particular action generally results in a request for additional information to allow the operator to specify the object to be acted on.

The lists of commands are described in files that are stored on disk, outside the object code of the Tools program. The name of the commands or command groups, the descriptive information presented to the user, and the names of other files associated with the commands are stored in these files. With this information stored outside the object program, additional commands and command groups can easily be added by simply changing the files, rather than by modifying the Tools program itself.

Figure 7 Tools application uses Action-Object



Assessing key constraints and decisions

In retrospect, our design for a Usability interface and for Installation/Configuration interfaces was profoundly affected by several early assumptions. Some of these assumptions were made without a great deal of analysis, because their impact on other areas of the design was not fully understood until after the project was complete.

Packaging. During the development of AIX, a major objective was to ensure that the operating system did not delay the hardware. To minimize the risk of any one AIX component's delaying the entire system, the components were managed as separate projects, with the intention of shipping only those that were ready when the hardware was ready. (In the actual event, all the components were ready when the hardware was ready.) Unfortunately, many of these components were not only managed independently, but they were also thought of as ultimately becoming separate Licensed Program Products (LPPs). Thus most of them had to be designed to operate independently of the other LPPs—cross-dependencies were not allowed. In particular, the Usability component (which includes Activity Management, the Files and Tools programs, and the dialog management subroutines that they use) was not considered part of the Base Operating System. Therefore, its interfaces could not be exploited by Installation,

Configuration, or other components that interacted with the user. This meant that Usability became yet another variant on the user interface, rather than the overall AIX interface, with the consequent reduction in transfer of user training from one task to another.

Supporting diverse configurations. The RT Personal Computer is a very versatile machine, capable of supporting a single-user workstation environment, a low-cost multi-user application, or both. Console displays range from the PC monochrome character-mode display to a family of megapel APA displays. The multi-user configurations can include both IBM and non-IBM Asynchronous terminals. The Usability interface designers took on the objective of building a single interface that could be used from any of these displays and terminals, with their diverse display capabilities and keyboards, and with and without a mouse. As a result, the Usability interface had a target lowest-common-denominator configuration that was very limited indeed. All mouse operations had to have a keyboard equivalent. Graphics icons were not possible because some of the displays were of the character-only type. The Usability interface was thus constrained to be an assistant for users of any of the configurations, at the possible cost of being a more powerful tool for users of the most sophisticated configurations.

The designers took on the objective of building a single interface that could be used from any displays and terminals.

Concluding remarks

Through our experience on the AIX part of the RT Personal Computer project, we have learned a number of lessons, and the experience has confirmed previous understandings in other cases. For one, system design can be strongly influenced by business decisions, such as the choice of applications, packaging, and supported configurations. Similarly, an accurate audience description is critical. A user interface perfectly designed for the wrong user audience is the wrong interface. Like programming bugs, usability problems must also be found and eradicated at the earliest possible stage of development. Problems that survive into the testing stages result in excessive development cost or reduced product usability. Early and intensive involvement of the human factors community in the design process results in substantially reduced breakage and a better final product. The human factors professionals are most effective when they are housed with the developers and architects during the design phase of a product development project. Collaborative problem resolution between human factors professionals and developers results in better fixes to more problems. Management support and commitment to incorporating human factors considerations in the design is imperative. Paper and pencil tests or intuitive judgments, even by professionals, are often misleading. A model of the user interface that can be tested with real subjects is essential to informed decision-making. Finally, we must always bear in mind that there is no detail too small to distract or frustrate a user. Thus, there is no detail too trivial to be assessed in human factors design. We believe that the development process we used has resulted in a superior interface for installation, configuration, and use of the AIX system. It successfully protects the novice from the UNIX complexities without limiting the UNIX expert.

Cited references

1. R. G. Bias and P. L. Smith-Kerker, "The mainstreamed human factors professional in the development of the IBM RT PC," *Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics*, New York (1986), pp. 153–158.
2. D. A. Norman, "The Trouble with UNIX," *Datamation* 27, No. 12, 139–150 (1981).
3. D. M. Ritchie and K. Thompson, "The UNIX time-sharing system," *Bell System Technical Journal* 57, No. 6, Part 2 (July–August 1978).

Frank Waters *IBM Entry Systems Division, 11400 Burnet Road, Austin, Texas 78758.* Mr. Waters is an advisory programmer. He has worked as a technical writer, programmer, and programming manager on projects such as the IBM 7040/7044, OS/360 Release 1, TERMTEXT, OS/VS1 and VS2, VSAM, 3850 Mass Storage System, 8100 Data Base and Transaction Management System, AIX user interface design, and RT PC product planning. He is currently engaged in advanced technology projects directed by IBM Fellow Glenn Henry.

Randolph G. Bias *IBM Entry Systems Division, 11400 Burnet Road, Austin, Texas 78758.* Dr. Bias is a staff human factors engineer. He received his Ph.D. in human experimental psychol-

ogy from the University of Texas at Austin in 1978. Prior to coming to IBM he taught at that university and served as a member of the technical staff at Bell Laboratories. In his three years with IBM he has supported the development of RT PC system software, AIX, and DisplayWrite 4. Dr. Bias is currently supporting the development of Operating System/2, Extended Edition. This support has taken the form of early usability testing and other efforts promoting software usability.

Penny Smith-Kerker *IBM Entry Systems Division, 11400 Burnet Road, Austin, Texas, 78758.* Ms. Smith-Kerker is a staff human factors engineer/psychologist. She has provided human factors support on projects such as keyboard design, training materials design, the IBM Displaywriter, AIX user interface design, customer surveys, and advanced technology projects directed by IBM Fellow Glenn Henry.

Reprint Order No. G321-5303.