This paper is a discussion of the rationale behind the design of the software user interface of the System/38. It presents the design approaches used to produce a highly usable interactive system. The three primary system user interfaces are also presented, showing how the approaches were used in their design.

The design rationale of the System/38 user interface

by J. H. Botterill

Although advancing technology is making systems that are more and more complex available to the users of small systems, the interface between the user and such systems cannot become correspondingly more complex. Instead, the interface must become easier to use so that more people and companies can take advantage of the richer function provided by the new technology. In the small-business environment, in particular, it is crucial that new systems be usable by the current staff of the business and not require the addition of new and sophisticated data processing expertise. Personnel costs already comprise 45 to 50 percent of most data processing budgets,[1] and thus it is important to minimize such costs.

In the past, much of the perceived ease of use of smaller systems was due to their limited function and to the fact that their users were primarily professional programmers, operators, and data entry clerks. These users were able to learn to use the system interfaces because they were trained as data processing personnel and the interfaces involved only a relatively few functions. Today, with the need for interactive systems and up-to-date data, both the personnel and the interfaces have changed. First, functional requirements have increased to include things like data base, communications, security, and workstation support. Second, more and more end users want to

© **Copyright** 1982 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

directly access and manage their own data as opposed to going through data processing personnel. Thus, the basis of the perceived ease of use of past small systems can no longer be used as the basis for new-system ease of use. New design approaches and standards are necessary.

Even for large and complex systems, traditionally programmed and managed by highly trained staffs, there has been a shift to a set of users less oriented to data processing, resulting in the need for breakthroughs in usability.[2] One reason is the high cost and limited availability of such highly trained people. Another reason is that owners of the data are no longer a part of the data processing organization and interface directly to the system through workstations.

This paper discusses the design rationale of the software user interface of the IBM System/38. It describes the ease-of-use requirements imposed by the interactive environment and how the System/38 user interface was designed to meet these requirements for increased usability. Here the term user interface is defined as the way the software communicates or interacts with the user to help in accomplishing his/her tasks. This interface then includes the means by which the system accepts requests from the user and the way information is returned to the user. Examples of types of input would be workstation display formats to accept requests and languages or formats in which requests can be expressed. Types of output include display formats, messages, and printer listings. The level of ease of use depends on what the user must learn and do to acquire the desired end results relating to his business.

the user requirements

Over the last six or seven years, systems have evolved from a pure batch environment to an interactive one. The System/38 was designed from its inception to be an interactive data-base-oriented system for the small-to-intermediate business environment. It is primarily a workstation system for a set of users designated as programmers, system operators, and end users.

Three broad user interface requirements had to be met for System/38 customers to reap the benefits of an interactive, on-line data base system:

1. Increase *programmer* productivity by providing tools and functions to assist in writing interactive applications and in converting batch applications to be interactive applications. Reduce the level of programming expertise required to program applications and manage the system. Give programmers convenient interactive access to system and utility functions so that the system function would be easier for them to use.
2. Provide the *system operator* with interactive facilities for easily managing the more dynamic environment of a workstation

system. Operators are faced with having to meet the input and output requirements of jobs they did not submit, and with which they may not be familiar. These requirements include special forms, diskettes, tapes, and output distribution, as well as the general backup and recovery of on-line data.

3. Increase ease of use for the *end user* who is not a data processing professional by giving the programmers what they need to conveniently produce easy-to-use applications for the end users. In addition, provide the end user a simple way to request applications, enter data, and request reports from the data base.

This paper addresses the user interface design approaches used in developing a system to meet these requirements, concentrating on general system-wide approaches and not specific approaches within a particular function. The emphasis is on how the user interface was intentionally designed so as to be easier to learn and use than systems with a comparable level of function. Rather than having a different interface for each type of function, with its own design approach and rationale, the System/38 has a coherent interface design across the entire set of system and utility function that is intended to be conceptually simple and consistent. Not described is another important element of designing usability into a computing system—the development process. The development process and controls used to ensure that the approaches and standards were adhered to are discussed in Reference 3.

Some of the design approaches followed were

- Using an object orientation
- Expressing functional requests in terms of a verb acting on an object
- Hiding the internals
- Minimizing the number of different interfaces and making them system-wide
- Ensuring a high degree of consistency within and between all interfaces
- Optimizing for the simple and normal

These and other approaches are described in the rest of the paper, followed by a brief discussion of the three primary user interfaces to the System/38, showing how the usability design approaches were used in their design.

General approaches

Object orientation

One of the approaches is the use of an object-oriented design. Objects are the means by which information is stored and processed. They are named collections of data and attributes that are visible at the user interface. The internal representation of the data and attributes is not visible.[4] The functions of the system operate on the external objects.

Prior systems have not been consistent in defining the visible entities within the system on which operations could be performed. They were at different levels and had little similarity in attributes. There were low-level entities like data control blocks and storage itself, medium-level entities like catalogs, and the higher-level entities like data files and programs. These entities were acted upon by low-level assembler and macro interfaces, a medium-level job control language, and higher-level utility and language interfaces.

On the System/38 all visible entities are high-level objects. They have an understandable external purpose and a set of useful attributes which can be set by their users. They can be operated on by a set of control language commands or by standard functions within the high-level programming languages.[4] The system manages the security and integrity of the objects and their content.

Objects are like furniture. There are different types of furniture that have different uses and characteristics, but all have fundamental similarities. If you know that an item is a piece of furniture, you know that it is movable and is used in a room, but you do not know its specific purpose or attributes. Knowing that it is a chair or table tells you those things. Similarly, knowing that something is an object identifies it as being in the system, that it can be accessed by name, and that it can be created, changed, moved, or deleted, among other things. As for knowing its specific purpose or attributes, you must know its type. The types of objects fall into one of the following four groups. If an object contains or allows access to data records, it is called a file. If it is invoked to perform processing, it is called a program. If it is descriptive, it is called a description. If it is a waiting line, it is called a queue. Table 1 lists these groups of objects along with examples of objects within each group.

The functions provided include some that are object-type-specific and others that are generic and operate on multiple object types. The object-type-specific functions primarily deal with the attributes of a particular object type. An example of an object-type-specific function is a create file function which defines a file and the attributes that pertain to a file. The generic functions operate on multiple types of objects. An example is the save object function that saves many types of objects.

Objects are brought into existence through a create command that defines the name, attributes, and initial value or values for the object being created. Each object is assigned a type that is determined by the object's specific purpose and corresponds to its create command, for example, Create Output Queue or Create COBOL Program. After an object is created, it remains on the system until it is explicitly

Table 1 Object grouping

Group	*Object type/subtype*	*Contents*
File	File	Data and data description
	Physical data base file	
	Logical data base file	
	Display file	
	Printer file	
	Tape file	
	Communications file	
Program	Program	Processing description
	Control Language program	
	RPG program	
	COBOL program	
Description		
	Device description	Device attributes
	Line description	Line attributes
	Subsystem description	Subsystem attributes
	Job description	Job attributes
	Edit description	Editing attributes
Queue		
	Job queue	Jobs
	Output queue	Output files
	Message queue	Messages

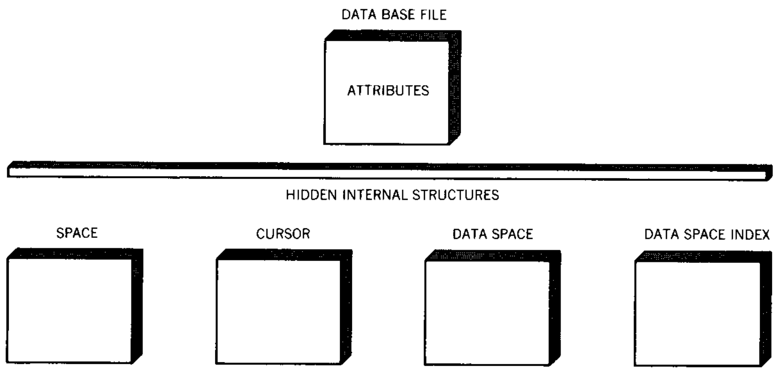
deleted by a delete command. During its existence, only operations that are valid for that type of object are allowed to be performed on the object. Only users who have authorization for the specific object and for the specific operations can perform them.

The key advantage of the object orientation is that the users only see and specify attributes that are meaningful externally. The internal structure and actual storage occupied by the information are hidden. Users do not have to know if a given object is implemented as multiple data structures or as one. They do not have to know nor can they get at the offsets or internal representation. For example, a data base file is made up of four machine object structures: a space, a cursor, a data space, and a data space index (see Figure 1). The system manages the individual pieces of the file in a way that allows users to perceive the file as a single object.

To minimize user learning, each type of object is designed similarly. Users need not start all over again to learn about a new object's design or use. Instead they can expect the new object to have a design similar to those with which they are already familiar.

This similarity relates to both the basic attributes of objects and the common operations that can be performed on them. Each object has a

Figure 1 Data base file object and internal structures



set of common attributes, including name, type, creation date, save date, restore date, and text description. Although object types differ, each is created with a similar Create command and deleted with a similar Delete command. Most object types can be changed with a similar Change command and displayed with a similar Display command. Most types of objects can be renamed, moved, saved, or restored using one set of commands which operate on multiple object types. Users can feel in control because the consistency makes them comfortable. They are certain that unknown things will function like known ones.

This object-oriented approach allows users to define workstation and printer devices to the system, create files, create application programs, and create job processing environments in a convenient, straightforward fashion. It gives the flexibility and extendability needed, in that one or more objects can be defined to meet the needs of each installation. Others can be added at any time.

Standard versions of all objects necessary for an operational system are shipped with the system. The initial or small system user does not need to create objects, such as job queues or output queues, to get started. The extendability and flexibility are available for when they are needed.

Verb-object function requests

Prior systems have had little consistency in naming or labeling functions such as commands, procedures, or macros. They have used a mixture of verbs alone, verbs followed by object name, objects followed by verb, nouns with verbs implied, and embedded adjectives and other secondary phrases that hide the base meaning of the name or label.

People want a computer system to *do* something for them. They are action-oriented, and they need to feel that the computer is there to help them and is under their direction. The user interaction needs to be designed with an "action against object" orientation to meet this need. Action requests, and text describing action requests, need to begin with a simple verb and be followed by the identification of the object of the action. Examples are Create Document, Clear Diskette, and Copy File.

Thomas and Carroll have studied the importance of hierarchy in producing a more usable command language.[5] Hierarchical command languages have multiple structural elements that are combined in a fixed way. A verb-object scheme is a dual-level hierarchy. Thomas and Carroll report that people rate hierarchically consistent command languages better than those that are not hierarchical. They found that people learn hierarchical command languages more quickly and that the frequency of some types of errors was reduced by using a hierarchical command language.

On System/38 the requests to perform operations on these objects come through a control language, interactive display responses, and command keys. The system-provided control language commands have names based on a verb-object hierarchy. A command exists in the Control Language for each function. Examples of commands of an operational nature are

- Start Diskette Reader
- Cancel Job
- Hold Job
- Display Active Jobs

Examples of commands of a programming nature are

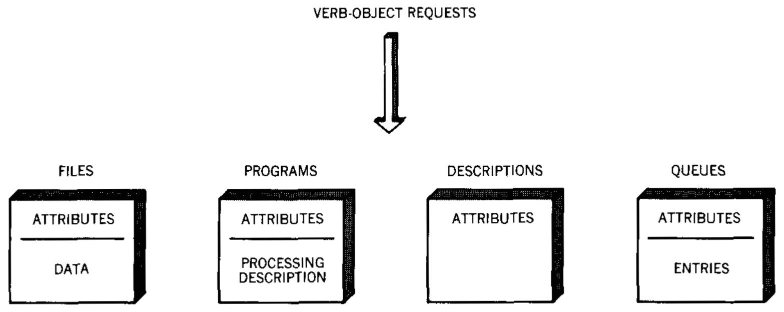
- Create COBOL Program
- Copy File
- Edit Source

Examples of commands of an end-user nature are

- Display Data
- Change Data
- Design Query
- Query Data

Where appropriate, options on the commands are also named using the verb and object approach. For example, on the Copy File command there is a parameter to specify whether or not to create the target file, and it is prompted on the display screen as "Create file?" The corresponding command keyword is CRTFILE(*YES or *NO). The keyword is formed by concatenating the abbreviation for create (CRT) and "FILE." The values are YES and NO with an asterisk prefix. The asterisk is used to distinguish the option values from user-defined names.

Figure 2 Verb-object design



The system-provided user menus are also verb-object oriented. For example, some of the options on the Program Call Menu are the following:

- Call program
- Display messages
- Send message

Some of the options on the programmer menu follow:

- Create object
- Submit job
- Display submitted jobs
- Edit source

Further details about these menus are provided later.

Command function (CF) keys supported on the display workstation can also request function. They are labeled with the actual command mnemonic which is a verb-object form or, if no command exists, by verb or verb-object text. For example:

CF6—DSPMSG	(Display Messages)
CF7—DSPSBS	(Display Subsystem)
CF3—Fold/Truncate	(The displayed data is the implied object)
CF5—Redisplay	(The displayed data is the implied object)

In these ways requests for function are designed to appear as a verb-object request against a set of high-level objects. Figure 2 illustrates this design.

Hide internals

In contrast to most prior systems, assemblers and internal dumps are not considered essential or desirable features for the System/38. Internal system implementation is hidden so that the user does not have to learn it. Needed function and information are provided to the user at the external interfaces in a way that meets the usability objectives described in this paper. The functions can be requested through a standard user interface such as the Control Language (CL), the Data Description Specifications, or the Interactive Displays. Therefore, the programmer does not need to know the internal data format to request a function using a low-level interface like a supervisor call. He/she does not have to request dumps or load maps to program or debug programs. A high-level debug facility is provided to allow the programmer to find problems by using a level of support equal to that used for writing the programs.

For example, a program named INVENTORY can be debugged by entering an Enter Debug command:

```
ENTDBG PGM(INVENTORY)
```

The programmer can request that the program, no matter whether it is written in CL, RPG (Report Program Generator language), or COBOL, stop at the statement labeled COMP by simply executing an Add Breakpoint command specifying a standard statement label within the program and the variables to be displayed, as shown:

```
ADDBKP STMT(COMPARE) PGMVAR(EMPNBR ACCT)
```

The contents of the variables in the program are displayed as

```
Variable: EMPNBR
  '333333'
Variable: ACCT
  '614-3614'
```

The contents of the variable ACCT can then be changed by pressing the CF3 key to get the command entry display and then keying in

```
CHGPGMVAR ACCT 316429
```

Execution of the program can then be restarted by keying in the Resume Breakpoint command:

```
RSMBKP
```

Information that is in other objects, such as a device description, can also be displayed or printed. It is returned in a form that can be used directly by the user or reentered to recreate the same situation or object.

In each case the internal structure and organization of the objects are hidden from the user's view. Information is made available in a form that can be used by the user. The user requests are against objects, not their internals or the system internals.

Minimize the number of user interfaces

In order to provide additional function, many previous systems proliferated the number of specialized user interfaces. Each procedure, facility, subsystem, or product had its own user interface. Such independence increased the complexity of the system at the user interface and significantly reduced the ease with which the user learned to use the function of the system.[6]

In the design of System/38 an explicit philosophy of minimizing the number of user interfaces was adopted in order to minimize user learning. The number of unique interfaces is minimized by having system-wide interfaces. A single control language, a single data description language, and a single interactive display interface are provided across the full set of system and utility functions. They are described in detail later in this paper. On small systems, the need for the consistency provided by a system-wide interface is very great because of role sharing. The operator and programmer or the programmer and data processing manager are often the same person. Even without this overlap, one user must often perform another user's functions (for example, a programmer performing operator functions). By presenting a consistent interface, artificial categories and boundaries are avoided, and learning is made easier. The primary interfaces used by each user type are shown in Figure 3. The programmer also uses high-level languages which are normally independent of the system.

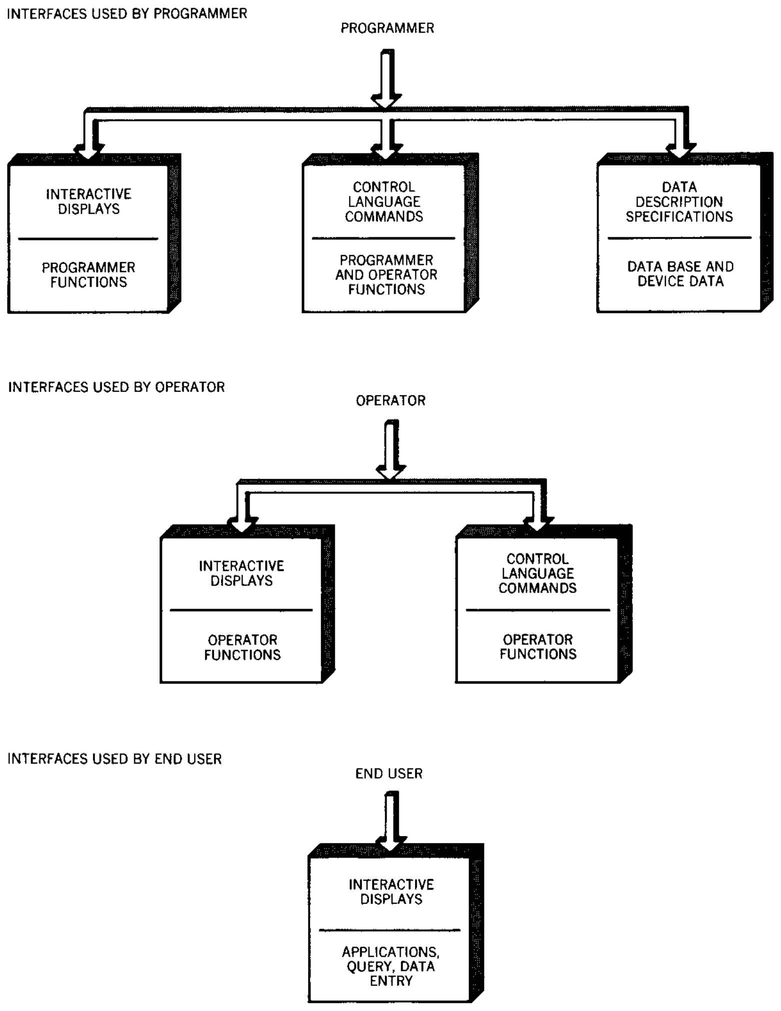
Grouping

Within each interface, related pieces of information or functions are named alike and grouped so as to help the user associate them and learn them as a group rather than having to learn them individually. Grouping (what psychologists call "chunking")[7] is the process of subsetting a larger whole. The user then has fewer items to remember. He/she remembers the groups that are fewer in number than the number of individual items. If like or related things are not named or presented as being alike, the person using them cannot benefit from the similarity or relationship and must learn each independently.

Examples of the use of grouping on System/38 are

- The command set of over 300 commands only uses about 65 command verbs. The verbs form groups of like commands. Ten of these verbs or groups account for two-thirds of the commands. For example, knowledge that a command is a display command identifies its purpose and how it operates. This verb groups some 50 commands that would otherwise need to be considered individually.
- RPG programs, COBOL programs, and CL programs are grouped under the object type *PGM. All can be called in the same way and debugged in the same way and are therefore perceived as part of a single set of like objects.
- Device files have three groups of attributes: spooling attributes, nonspooling attributes, and common attributes. The spooling

Figure 3 Minimum number of system interfaces


INTERFACES USED BY PROGRAMMER
PROGRAMMER
INTERACTIVE DISPLAYS
PROGRAMMER FUNCTIONS
CONTROL LANGUAGE COMMANDS
PROGRAMMER AND OPERATOR FUNCTIONS
DATA DESCRIPTION SPECIFICATIONS
DATA BASE AND DEVICE DATA
INTERFACES USED BY OPERATOR
OPERATOR
INTERACTIVE DISPLAYS
OPERATOR FUNCTIONS
CONTROL LANGUAGE COMMANDS
OPERATOR FUNCTIONS
INTERFACES USED BY END USER
END USER
INTERACTIVE DISPLAYS
APPLICATIONS, QUERY, DATA ENTRY

attributes are valid at certain times and for certain types of files. If the type or time of a desired operation is known, one or more sets of attributes do not even have to be considered.

- Information on listings is grouped under subsection headings rather than having a single continuous listing. This grouping helps to locate specific information.
- Like parameters are grouped on commands, such as target identification parameters, attribute parameters, and special option parameters.
- Functions are grouped via user menus, for example, the System Operator Menu and the Programmer Menu. This grouping not only helps to locate the desired option on the menus but also structures the function on the system so that any user can see an overview of what functions are provided.

High degree of consistency

Along with the diversity of interfaces, prior system user interfaces have been characterized by inconsistency. Inconsistency is the natural outcome without a strong, explicit, well-managed effort. A strong attempt has been made to make System/38 interfaces consistent. Each interface is designed to be consistent within itself in every way possible, including use, operation, naming, syntax, formatting, ordering, grouping, and editing.

This consistency is carried across system interfaces where appropriate. It includes the level of function (create, change, rename), terminology, ordering, and defaulting. If the interfaces are of a similar type, the consistency goes much further. For example, if two interfaces use a keyword syntax, the values and keywords for like options are alike. Similarly, like information presented on multiple output interfaces is consistent. For example, the parameter order in command input, in prompting, in publications, in display presentation, and in listing presentation is consistent.

IBM's customers have greatly appreciated the consistency and in fact rely on it. They therefore are quick to point out inconsistencies or missing functions (e.g., a missing change command) that they have recognized because of the prevailing consistency. Some of the cases of missing functions are the result of normal development resource constraints that are present in the development of any new system. Each new release fills some of these perceived deficiencies and corrects inconsistencies that are changeable without impacting operational compatibility for existing users.

The emphasis on consistency has definitely proven to be worthwhile. The way users rely on it and the concern for exceptions point out the need for continued attention to this design principle.

Early validity checking

Any time a user enters a request at a workstation that will be processed at a later time, validity checking of the request becomes very important. It should be unnecessary for the user to wait until the time of processing to be told about an error in his/her request. If the error is identified when a request is entered at a System/38 workstation, messages are returned to the display where the invalid values were entered. The user can then immediately correct erroneous values while the prompt text, the list of valid values, and help text are available. Errors are not ignored and values are not changed without informing the user. All users can get what they want and know what they get.

In addition to saving time, early and stringent validity checking results in

- *Improved learning.* Users are informed immediately when they make errors. Misconceptions are straightened out immediately, not reinforced. Users learn by trying because they receive immediate feedback.
- *Reduced worry and uncertainty.* Users always know whether a function will be performed and have a high degree of confidence that it will be performed correctly.
- *Straightforward understanding of what is used.* A specified value is not ignored or another value is not used without the definer being aware of what is happening.
- *Easier problem determination.* Errors are diagnosed immediately where they are made. The users are made aware of errors while they still understand the context, their own intent, and their alternatives. Errors do not go unnoticed until the consequences are great and both problem determination and error recovery are more difficult.
- *A more reliable system.* If an object is created, that object is correct. This principle reduces the danger of unexpected failure later if new, data-dependent paths are taken.

An attempt has been made on the System/38 to check the validity of all input. For example, when a query is being defined, the specifications are validity-checked. When programming language input is being entered, it is validity-checked to ensure later compilation. When commands are being entered to define a batch job, they can be optionally validity-checked to ensure that the job will not be rejected at execution time. This aspect will be discussed in more detail later as it pertains to the Control Language.

Optimize for simple and normal

An attempt was made to design the commands, displays, and listings so that commonly used options or attributes are shown first and so that specialized or less frequently used options or information are only seen when needed. All options or attributes beyond the simple base set are defined as optional choices.

This design minimizes the effort required to perform common functions or find commonly needed information. Several specific ways in which this principle has been applied on the System/38 are described below:

- *Summary listings* are provided by default, with options on the command to get more detail. For instance, the Display Object Description command defaults to one line of information per object. The user can request a full listing of object attributes.
- *Summary displays* are provided by default with one-line entries for each item. An option can be entered in front of any entry to request more detail. For instance, when the operator displays the list of the spooled output files for a particular job (as shown later

in Figure 13), he/she can display the contents of one or more of the files by keying in a one next to them or can request a display of detailed attributes for one or more of the files by keying in a two next to them. Other normally used operations, like hold, release, and cancel, are supported by entering other option numbers.

- *Basic parameters or values* are shown first on command prompt displays, presentation displays, and listings. For instance, on a display command with an option to request the basic or full set of information, the "full" option results in an output format where the "basic" information appears first followed by the additional information.
- *Simple commands* are provided for the simple base function. For example, a command is provided to create source files even though they can be created by a more general Create Physical File command. When using the Create Source Physical File command, the user only needs to enter the following to create a file of name SOURCE1:

  ```
  CRTSRCPF FILE(SOURCE1)
  ```

  Using the Create Physical File command, the user would need to enter the following:

  ```
  CRTPF  FILE(SOURCE1)
         RCDLEN(92)
         FILETYPE(*SRC)
         SIZE(20 20 499)
         ALLOCATE(*NO)
  ```

Changeable attributes

Object and environment attributes need to be easily changeable. The user does not appreciate having to recreate an object to change one attribute. On System/38, a change command is provided for most types of objects to allow the attributes to be easily changed. Within the interactive interface, a command function key is supported on important attribute displays, such as Display Spooled File Attributes and Display Device Status, to request the prompt display for changing the attributes. Having ways to conveniently change attributes allows the programmer and operator to capitalize on the defaulting approach and to learn by doing.

Match user level

It is important that the amount of information and assistance given by the system match the amount needed. If the assistance provided is more than needed, use of the function will be laborious. If too little assistance is provided, the user may quickly become frustrated and lost. It is therefore important that system interfaces be designed for their users. If a user needs help beyond that initially shown, he/she should be able to request more detailed information.

On System/38, formatted input prompt displays are used to describe the values or options that can be entered and to let the user enter values without having to use a rigid syntax. These prompt displays are only shown when the user requests prompting assistance. This results in the tailoring of the interface to his/her needs. Prompting is described in more detail under the Control Language.

Another way the interface is matched to the user is via the user profile that is part of the security support. One of the entries in that profile can be the name of an initial program to invoke when the user signs on to a workstation. By using this capability, whenever a user signs on, he/she automatically sees the first display for the application or function each normally uses. This capability is useful for almost all users and allows the tailoring of the interface for each user.

Yet another way of matching the user's needs is to make available short cuts, such as answer-ahead on menus. This allows the user to bypass the next menu when he/she knows ahead of time what he/she wants to do and how to request it. For example, the Programmer Menu allows the user to bypass subsequent menus for Query and Data File Utility data entry.

Forgiving of user

When errors are detected, the user needs a consistent, easy way to correct the errors, to back up, or to exit. The user should not be put in a position where he/she cannot recover or determine how to recover from an error. Errors should be considered normal, and the system should treat them as such.[8] Sufficient instructions, in the user's terms, need to be available.

On System/38, the normal approach to error correction on the workstation is to reverse-image the values in error, position the cursor at the first erroneous value, and display an error message for each error. The reverse image provides for rapid identification of which values are wrong. The cursor sets up the workstation for easy correction. The message gives a description of the error. By pressing the HELP key with the cursor on the message for which help is desired, additional information about the error can be received. The user can change any of the input values and then press ENTER to have them validity-checked again. Command key one is always available to request an exit.

Another area of forgiveness is in accepting input in more than one way. The system should not require the user to conform to one arbitrary way. Examples of this area on System/38 are tolerance of uppercase and lowercase for names where uppercase and lowercase distinction has no meaning, tolerance of the presence or absence of leading zeros for a numeric value, and allowing either the presence or absence of apostrophes around a character string with no embedded blanks. Distinctions in these cases would be viewed by users as unnecessary and frustrating.

Optical device features

The system is designed to make use of advanced device features even when they are optional. If the optional features are installed, the system uses them. This design aspect contrasts with the common philosophy of having the system only make use of standard device features so that the system looks the same to all users.

Some examples of how the use of advanced features improves the user interface follow. On the majority of devices that support lowercase, the output text and messages appear in lowercase. On uppercase-only devices they appear uppercase. On display devices that support reverse image, input fields that are in error are made to stand out for quick correction by showing them in reverse image. On devices without the reverse-image capability, the user must roll through the messages and identify the fields in error by the text of the messages.

Another display feature that varies between display devices is the number of display lines. Display output is adjusted to make use of the size of the display workstation being used. More entries on a list are shown on a larger display.

Interfaces

In order to show what these general approaches mean and how they were applied to the design of the System/38 in practice, we will discuss the three primary user interfaces to the system. In addition, other approaches that are unique to individual interfaces are described. All function requests, except for those that go through the high-level programming languages common to most systems, are entered through one of three interfaces:

1. Control Language
2. Data Description Specifications
3. Interactive Displays

Control Language

The Control Language (CL) is the single command-level interface to the system. It includes commands for configuration, job control, operation, programming, query, and object management, to name just a few. It is primarily for the programmer and operator. CL supports the interactive and batch request of functions. Almost all commands can be used either in batch or interactively without changing the way they are coded. It supports full screen input and output to the display workstation, arithmetic functions, and application requests. CL is, in many ways, a high-level language for performing system functions and application control. It can be compiled for more efficient performance and supports variables of three data types: character, decimal, and logical.

syntax

As described in the article *The Rule-Driven Control Language in System/38*,[9] the basic syntax of CL is simple and free-form. CL uses the blank as the separator because it is a natural separator that is common to all countries, unlike the comma, which is used as the decimal point in many countries. The command name and associated parameters can begin anywhere on the record, thus allowing indentation and parameter alignment. Each parameter has an associated keyword that can be used to identify the parameter value. The keywords may be omitted for the first set of parameters, and only values need be entered if the user enters the values in a fixed positional order. For example, the Copy File command is defined to have the following form:

```
CPYF   FROMFILE (file-name) TOFILE(file-name) . . .
```

A request to copy File A to File B can be coded with keywords in either of the following two ways:

```
CPYF   FROMFILE(A) TOFILE(B)
CPYF   TOFILE(B) FROMFILE(A)
```

The keywords can be coded in either order because the keywords identify the values A and B. The same request can be coded positionally without keywords. Then the values must be coded in the order of the command definition shown above. For example:

```
CPYF   A B
```

command naming

The command names for system functions consist of verb-object pairs made up from a small set of primarily three-character abbreviations. In the above example, "CPY" is the fixed abbreviation for the verb "Copy," and "F" is the fixed abbreviation for the object "File." The rule scheme used to generate names throughout the system, including the mnemonic name of command, keyword, and object names, is to concatenate the abbreviations of each word in the descriptive name. For example:

Create User Profile—CRT + USR + PRF = CRTUSRPRF
Delete User Profile—DLT + USR + PRF = DLTUSRPRF

Abbreviations other than the last one follow a rule scheme of taking three representative letters from the word. These include the first letter followed by two consonants. The consonants chosen are those that are the most prominent in the pronunciation and those that are most apt to distinguish the word from others. Other than an initial vowel, vowels are not normally included. This is done to avoid abbreviations that form a word in another language, possibly even a word with an unacceptable meaning. Below are examples of abbreviations made up of three characters:

Start	STR
Diskette	DKT
Reader	RDR

By having abbreviations of a fixed length, it is possible for the control language user to easily parse a name formed from multiple abbreviations and determine its meaning. For example, STRDKTRDR can be recognized as STR + DKT + RDR, which is the command to Start Diskette Reader. Without having such a rule, the command could be interpreted as being formed from ST + RDK + TRDR or S + TRDK + TR + DR.

The last abbreviation in a name is sometimes reduced to less than three characters to minimize keying. But it is only done if it can be done consistently and does not produce any ambiguity in parsing the abbreviations. For example, description is abbreviated D because the object types are called descriptions and the word is always last, as seen below:

DSPDEVD	DSP + DEV + D	Display Device Description
CRTJOBD	CRT + JOB + D	Create Job Description
CHGSBSD	CHG + SBS + D	Change Subsystem Description

Three-character abbreviations are used because two characters are insufficient for uniqueness without using characters that do not relate to the word being abbreviated. The set of abbreviations for words across the system beginning with IN shown below illustrate how three characters provide for sufficient uniqueness.

Initial	INL
Initialize	INZ
Integer	INT
Invocation	INV
Interval	ITV

Using more than three characters results in names that are too long.

In a few cases, exceptions to the vowel rule were made because of the strong precedence of common-use abbreviations that themselves were already three characters. Examples are LIB for library, REF for reference, REL for relation, and DUP for duplicate.

The examples in Table 2 describe this naming strategy more fully as it applies to command names. The approach has proven to be very extendable and rememberable. Users have found that they can readily learn the names because the consistency is strict and the command coding seems very natural.

keyword and value naming

Keyword names and value names frequently identify an object or option and not an action. A single unabbreviated word is used for this type of name wherever possible. For example, FILE, TYPE, OUTPUT,

Table 2 System/38 naming strategy

Examples of abbreviations used in command names

Verbs	*Objects*			
Create— CRT	Command—	CMD	Description—	D
Change— CHG	Device—	DEV	File—	F
Delete— DLT	Diskette—	DKT	Queue—	Q
Display— DSP	Display—	DSP		
	Message—	MSG		
	Program—	PGM		

Examples of command names

Create Device Description	CRT + DEV + D = CRTDEVD
Create Display File	CRT + DSP + F = CRTDSPF
Change Message Queue	CHG + MSG + Q = CHGMSGQ
Delete Message Queue	DLT + MSG + Q = DLTMSGQ
Display Message File	DSP + MSG + F = DSPMSGF
Delete Program	DLT + PGM = DLTPGM

and TEXT are common keywords and *YES, *NO, *ALL, and *NONE are common values. If the keyword or value name represents a multiple-word phrase, like source file, the rule scheme for concatenating three-character abbreviations is used. In either case the values are defined so as to be self-documenting and nonambiguous. Values may be numeric, character strings, names, or special values.

Special values are the identification of defined command options. They are always preceded by an asterisk so that they are not confused with names of objects or so that they do not preclude the use of that same identification as the name of an object. For example, in a source member keyword (SRCMBR), a special value, *PGM, is supported to mean the use of the source member whose name is the same as the program being created. The name of the program being created is specified in the PGM keyword, so the special value is named *PGM. The keyword then has the following definition:

SRCMBR(*PGM or source-member-name)

The asterisk avoids having to preclude the specification of a source member name of PGM. Another example is a file keyword with a special value of *ALL representing the option of using all files, not just one. This keyword has the following definition:

FILE(*ALL or file-name)

The file name specified could be any file name, including ALL.

By consistently prefixing all special values, the user does not have to decide whether a parameter requires the asterisk or not. Parameters that today only support special values, such as *ALL or *NONE, can later be extended to support a user-specified object name.

This approach to naming results in consistent self-documenting keywords and values without reserved name restrictions that would be error-prone.

command description

Each individual command is described to the system by the use of a Create Command function. The detailed description of each command and its parameters is stored in a command description object.[9] The object serves as a rule so that the command can have its validity checked by a single command analyzer and interactively prompted for by a single command prompter. The information includes:

- Name of the command
- Description of each parameter including keyword name, valid values, prompt text, and a default value to be used if a value is not specified for the parameter
- Name of the program to perform the function
- Identification of when the command is valid: interactive and/or compiled environment

A Display Command function is provided to display the primary attributes of a command description object. The prompter presents the parameters and the acceptable values for them.

The Create Command is provided to allow a user to create commands to invoke his own programs. These programs can be CL programs of one or more other commands or high-level language application programs. It allows the user to have the full benefit of the parameter validity-checking, prompting, and defaulting facilities. In this way, the customer can extend the command set to include personal commands to invoke system-related functions or application programs. A study at Bell Laboratories indicates that allowing the users to define their own commands is probably the only way to have command names that are natural for more than 40 percent of the users of a system.[10]

validity checking

As was discussed previously, early validity checking is a system-wide strategy. On prior systems, control language validity checking of syntax and values has usually been done at execution time prior to actual data processing. In some cases, an early check of some type of source has been done at source entry time. It has usually been a separate checker, covering only noncommand, unique syntactical errors like missing commas and unmatched parentheses.

On System/38 the command analyzer has the benefit of a command description object which contains the full description of the command necessary to do a thorough validity check.[9] It can report errors in keyword names, values, value type and value length, and interparameter value conflicts. The validity checking is performed at command execution time when a command is entered at a workstation, is executed within a batch job, or is executed within a CL program.

It is performed during interactive command prompting as individual groups of parameters are entered in response to system prompting displays in either the source entry or execution environments.

It is performed at source entry time as commands are entered through the source entry function to be put in a data base file for later compilation into a CL program. Similar checking is done during the compilation.

A job option exists to have the CL commands validity-checked as the job is placed on the job queue for later batch execution.

Because the validity checking is always performed by the same command analyzer, based on the same command descriptions, the user receives the same messages in each case.

parameter defaulting

The System/38 control language utilizes a new, highly visible defaulting approach.[9] Most parameters are defined as optional. Each optional parameter is defined with a carefully selected default. This default is the value that is used if a value is not specified for the parameter. Defaults are selected based on the most commonly used value.

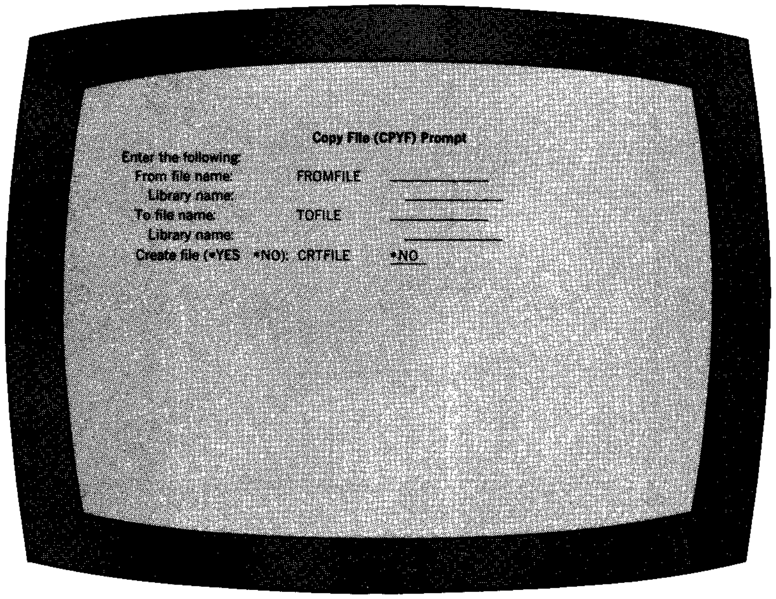
Using an approach with many defaults rather than requiring the user to specify a value for every parameter helps the user by requiring less keying and less knowledge. It requires less keying because only special attributes need to be specified.

Less knowledge is required because users do not need to understand all values available in order to choose one to perform the desired task. They can let the parameter default and get the function up and running. As learning progresses, they can change the values to tailor them to meet special needs. For example, an output queue can be created, and the number of job separators be allowed to default to one. Later, if it is found that three would be better, the number can be changed.

Many systems have used some form of defaulting approach. Several problems have been common in these implementations that have detracted from the strategy meeting the objectives stated above.

The first problem has been that the default taken is not what was expected. On System/38, much of the mystery (what gets defaulted, and when) is avoided by making all defaulting visible and well-defined. The default value is one of the standard user-specifiable

Figure 4 Example of command prompt



```
                         Copy File (CPYF) Prompt
Enter the following:
  From file name:          FROMFILE    ____________
    Library name:                        ____________
  To file name:            TOFILE      ____________
    Library name:                        ____________
  Create file (*YES  *NO): CRTFILE     *NO
```

values and therefore is fully described in the supporting documentation. Wherever appropriate, values are named with a specifiable word value that is descriptive of its meaning, for example, *NORMAL for an authorization default, or *NOLIMIT for a file-size default. Such self-descriptive values are defined in the command description object, documented in the control language reference publication, and shown on the prompt display for the command, as shown in Figure 4.

Use has shown that displaying the defaults is crucial because it allows the user to see the default values and become accustomed to their being consistent. Some users have difficulty ignoring parameters they do not understand. Displaying the defaults allows a decision to be made prior to execution based on the defaults, not after execution.

On systems where the default is not required to be a specifiable value, the default action tends to develop its own idiosyncrasies even though it was intended to be like one of the standard values. The result is more mystery, more options, and the inability to specify the default action when it is what is wanted. On System/38, the default value may be optionally specified. If the default is specified, the same action is performed as if a value were not specified. Specifying the value provides documentation to any person reviewing the program or command log. A strong effort was made to have uniform defaulting. Similar functions with similar options default to the same value. Thus, the defaults are easier to remember, and the need to refer to documentation is reduced. A formal benefit of having specifiable defaults is that a value may be specified without concern for whether or not it is the default. This benefit is important when the user knows what he wants; the simplest thing for him to do is to just specify it.

Another problem that often accompanies defaulting is receiving error messages as a result of the defaults. To minimize this situation, the defaults in a set for a given function are carefully chosen so as to be compatible with one another, resulting in a consistent, meaningful action instead of error messages. In those few cases where there are strong relationships between values for two parameters, a conditional default is defined so that the parameter will default properly based on the values specified on the related parameter.

parameter prompting

On many systems no interactive command prompting is provided. Some of these systems have on-line help text that can be requested at any time. Although the help text reduces the need for use of publications, it still requires the user to determine how to enter the command using commas, parentheses, or other syntactical delimiters. If the help text displays do not allow keying in of the command on the help screens, the user must remember the instructions in the transition back to the input display. On other systems, the prompting results in only one parameter value at a time and provides no way to determine the valid values that can be entered. On still others, the prompting is only available after the user makes an error.

As part of the System/38 interactive display interface, which is discussed in detail later, a system command prompter is available to assist the CL user in entering commands. The prompting can be requested by pressing a command function key while entering the command. The same prompting is provided when entering a command interactively for immediate execution or entering a command as part of a CL program for later use. The prompter identifies parameters, defaults, and valid values so that the user can enter commands without frequent reference to publications. The list of valid values can be requested by keying in a question mark in the field of interest. All the information necessary for this assistance is obtained from the command description object.

Prompting can be requested at almost any time during the keying of a command. The user can key in however much he/she knows and ask for prompting if and when the need arises. In this way the interface adapts to match the user level. The assistance is available for the user needing it, but the assistance is not a frustration to the user not needing it. The prompting uses however many lines are on the screen of the device that is displaying descriptions and input fields for multiple parameters at a time. Optional device features such as reverse image are utilized to assist the user in error correction. Figure 4 shows a prompt display for three parameters of the Copy File command. Beginning at line 3 there is a line per value with the text description of the parameter followed by the keyword name, followed by the input field for the value.

Each input field is the length of the longest value supported for that parameter. If the parameter has a default value, it is shown in the input field. Defaults are named so as to be descriptive of the option they represent. The CRTFILE parameter is shown in the example with a default of *NO. That value can be accepted or keyed over with the other valid value of *YES. The user can quickly review the command parameters and their defaults and key in only the values that he wants to change. (This is illustrated later in Figure 7.) In this way the user is freed from having to specify parameter names or having to adhere to special positional or syntactical requirements.

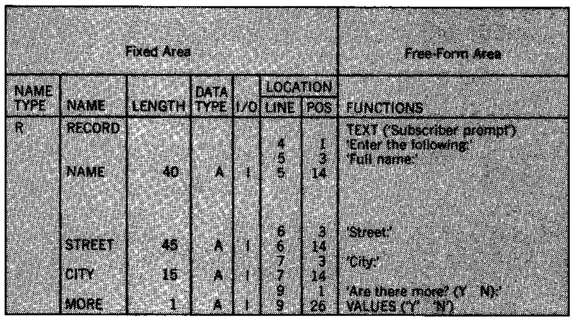
After the values have been keyed in, they are validity-checked, and immediate feedback is given. In order to allow for easy error correction, the display is reshown with the values in error reverse-imaged, the error messages at the bottom of the display, the cursor on the first value in error, and the keyboard unlocked ready for the user to key in the correction. The user can correct one or more of the erroneous values or change any other values and have those values checked again. This process continues until the user and the system agree that the command is ready for execution or entry into a source file.

In summary, the System/38 CL is designed to be a user interface separate from the programs that provide the function. It is designed to have a minimum of syntactical rules; thus, the commands themselves exhibit a high degree of consistency. When assistance is needed, a command prompter is available to provide assistance in entry of the command as well as to provide immediate feedback to allow correction of incorrect specification. Surveys of end users and their evaluation of System/38 usability indicate that both programmers and operators are happy with the ease of learning of the system commands and with the ease with which they can be entered. The prompting and consistency are almost always identified as the major reasons for the System/38 CL being rated easy to use by a majority of users surveyed.

Data Description Specifications

On previous systems, data in external storage has been defined by the program accessing the data (input specifications in RPG II, DECLARE statements in PL/I). Data on the System/38, in contrast, can be externally described—that is, the description of files and formats is external to the using program. It is stored in the file object. Just as there is a single CL control interface to the System/38, there is a single data description interface. This data description is accomplished via Data Description Specifications (DDS) written by the application programmer or, in the case of display formats, defined indirectly by simply laying out the display format interactively, using the Screen Design Aid. DDS allows him/her to describe each file, each record in a file, each field in a record, and the order of the fields.

Figure 5 Example of DDS source statement



Fixed Area							Free-Form Area
NAME TYPE	NAME	LENGTH	DATA TYPE	I/O	LOCATION LINE	LOCATION POS	FUNCTIONS
R	RECORD						TEXT ('Subscriber prompt')
					4	1	'Enter the following:'
					5	3	'Full name:'
	NAME	40	A	I	5	14	
					6	3	'Street:'
	STREET	45	A	I	6	14	
					7	3	'City:'
	CITY	15	A	I	7	14	
					9	1	'Are there more? (Y N):'
	MORE	1	A	I	9	26	VALUES ('Y' 'N')

These descriptions are stored within the file when it is created. DDS is used to create physical files (containing data), alternative views of that data (logical files), display formats (display files), printer output formats (printer files), and communication formats (communication files).

The DDS source statement consists of a fixed-format area for the most common specifications (e.g., field name, length, and data type) and a free-form area for less-used specifications. Figure 5 shows the primary fields and the basic form layout.

This type of format was chosen because it is very similar to the way users document fields in records today. It is similar to the specification of a set of declare statements in many programming languages and directly corresponds to the RPG input and output specification form approach with which the vast majority of System/38 users are familiar.

The programmer simply lists the fields, one per line, specifying the name, length, and data type. A minimum amount of coding is required, and the result can be easily read.

A variety of special processing options and device-dependent attributes can be specified in the free-form area. They are specified with keywords or keywords with values, which follow many of the same syntactical rules as CL. In this way the user does not need to learn a new approach to specifying free-form information.

DDS source statements are normally entered into a data base file member using the Source Entry Utility (SEU), or defined indirectly by using the Screen Design Aid. In SEU, prompting and immediate validity checking are provided. As with the prompter, the display is set up for easy error correction.

Several benefits result from separating data descriptions and programs:

- There is a reduction in the amount of coding required in the high-level language. The data description is coded only once, for the file. The Data Description Specifications for display files can include validity-checking parameters as well, thus removing the need for validity-checking code in every program that uses the file. It also results in system-provided early validity checking of data entered through system or application displays.
- A high degree of data independence is achieved. Changes to data attributes or the addition of new fields to a file do not necessarily require the recompilation of the using program.
- The user has greater control over the naming and defining of data. It is easier, for instance, to implement installation-wide naming conventions.
- Application documentation is improved. The user can define a text description for each file, each record type in a file, and each field in a record. This text is stored in the file description that is a part of each file.
- The owner of a file can secure specific sensitive fields by defining a logical file that provides only the desired data, and then authorize only the use of that logical file.
- Record lengths and other file layout considerations are no longer important design considerations. The specific character position of a field in a record is not specified nor is the record length. Different record types with different lengths may be in the same logical file.
- The logic of the application can be addressed separately from the structure of the data.

These benefits are made available to the high-level language programmer. RPG and COBOL have been naturally extended to make use of externally described files and to utilize system data base and device data management through the standard language input/output constructs. This provides for simpler and more straightforward programming than the approach of having to use external calls to system-provided subroutines.

Interactive Displays

The interactive display interface provides a set of workstation displays that allows the interactive user, who may be a programmer, operator, or end user, to request functions and information to meet his/her needs.

The displays are designed to be easy to learn and use, and every attempt was made to follow the best available display design guidelines.[11-13]

Each display is clearly titled on line 1 to identify its purpose and confirm that the desired display was received. If the display requires user action, such as selecting an option or entering values to define a request, this is communicated by instructional text in line 2. The body of the display is dedicated to the main purpose of the display, which may be to present a menu of choices, a series of input prompts, or output information. Messages are always displayed at the bottom. Lowercase text is used to improve readability.

The interactive display interface is designed to require less user knowledge and keying than the Control Language and Data Description Specifications. A separate input field is normally provided for each input value; therefore, syntactical delimiters and keywords are not needed to separate, delimit, or identify the values. The text preceding each input field identifies the type of information the user should enter in that field. Input fields are underlined to identify their location and show the maximum length of the values accepted.

four display types

Almost all displays fall into one of four basic types of display, each designed for a purpose. The consistency across displays makes the set of displays for all system and utility functions form a single interface that operates under a single set of rules covering format, command keys, messages, operation, and design philosophy. Thus, by learning how to operate these four types of display, the user is able to operate any one of the over one hundred system displays. The four types are described below.

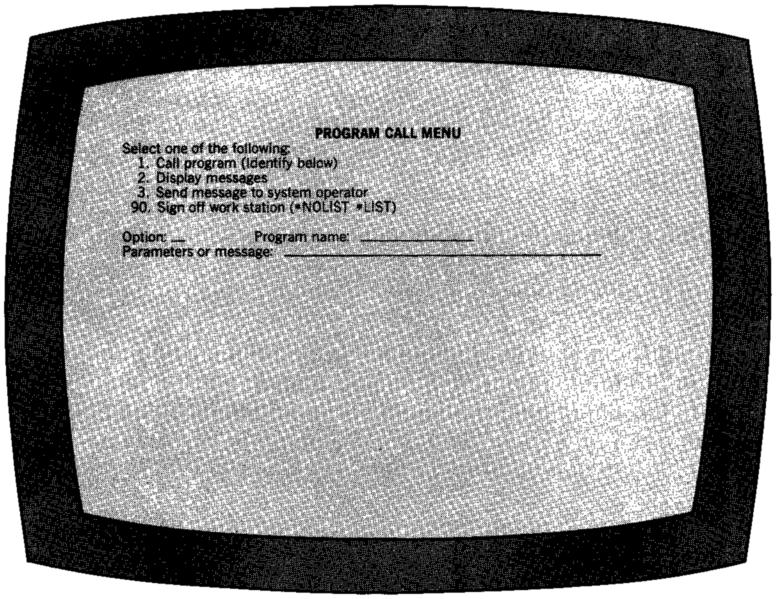
A *menu* allows the user to select a function from multiple alternatives. Examples of menu formats are shown in Figure 6.

A *prompt* requests one or more values to be entered in a simple fill-in-the-blank format with each input field preceded by text describing the value to be entered. It is shown in Figure 7. The first display is a command prompt and the second display is a query output prompt. The input fields often contain a default value that is used if another value is not keyed in. These defaults minimize the need for values being keyed in and allow most requests to be entered by keying in only a few values. They are designed to be self-explanatory, for example, *ALL or *NONE. The locations of input fields and their lengths are clearly shown by being underlined, which corresponds to the familiar fill-in-the-blank technique.

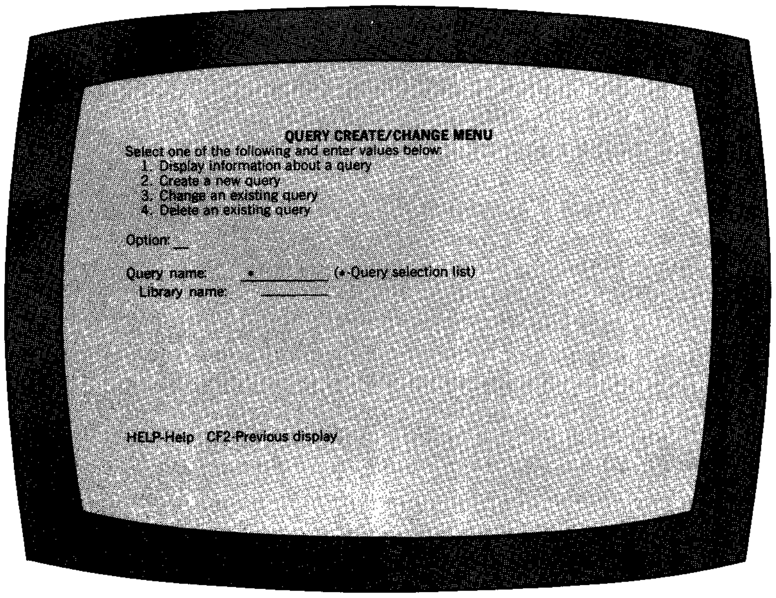
A *columnar selection display* presents multiple entries, each of which includes the name of the entity represented and its key identifying attributes. This display type is shown in Figure 8.

This type is a keystone of interactive ease of use of the System/38. A one-character input field is provided in front of each entry. By simply entering an option number in front of one or more entries, the operation represented by that option is performed on the entities. The requesting of functions by keying in only a single digit is thus made possible. The location of the entered number completely identifies the target. The valid options are described below the presentation area.

Figure 6 Examples of menu displays



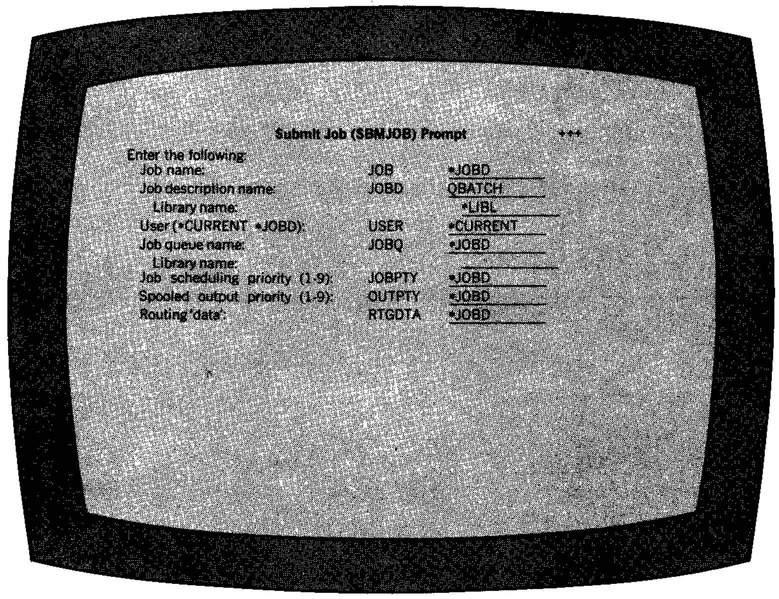
```
                    PROGRAM CALL MENU
Select one of the following:
  1. Call program (identify below)
  2. Display messages
  3. Send message to system operator
 90. Sign off work station (*NOLIST *LIST)

Option: _        Program name: ____________
Parameters or message: ____________________________________
```



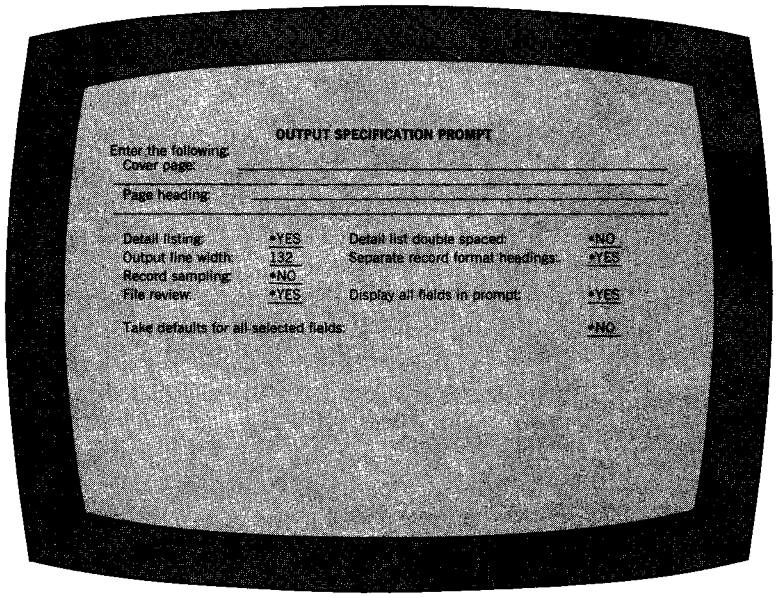
```
                 QUERY CREATE/CHANGE MENU
Select one of the following and enter values below:
  1. Display information about a query
  2. Create a new query
  3. Change an existing query
  4. Delete an existing query

Option: __

Query name:      *_________ (*-Query selection list)
  Library name:   _________



HELP-Help  CF2-Previous display
```

The approach allows any combination of operations to be requested against an almost unlimited set of targets on one interrupt. The keys labeled Roll Up and Roll Down are supported, allowing the user to view all available entries and enter options for any of them before asking that they be performed.

Figure 7 Examples of prompt displays



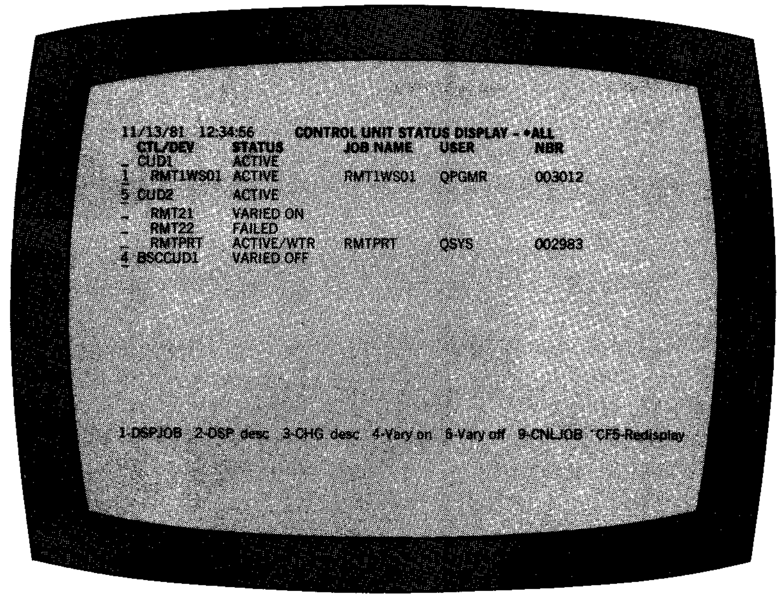
```
                    Submit Job (SBMJOB) Prompt              +++
Enter the following:
  Job name:                        JOB       *JOBD
  Job description name:            JOBD      QBATCH
    Library name:                              *LIBL
  User (*CURRENT  *JOBD):          USER      *CURRENT
  Job queue name:                  JOBQ      *JOBD
    Library name:
  Job scheduling priority (1-9):   JOBPTY    *JOBD
  Spooled output priority (1-9):   OUTPTY    *JOBD
  Routing 'data':                  RTGDTA    *JOBD
```



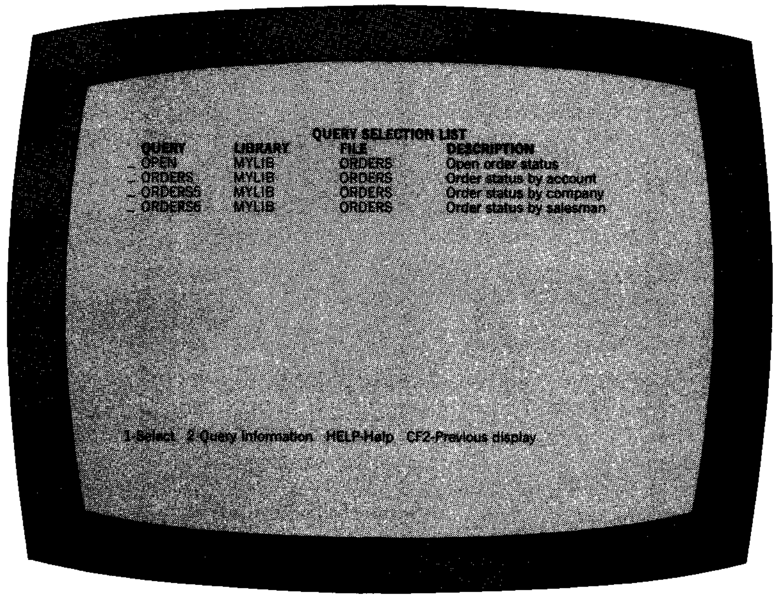
```
                 OUTPUT SPECIFICATION PROMPT
Enter the following:
  Cover page:
  Page heading:

  Detail listing:       *YES   Detail list double spaced:         *NO
  Output line width:    132    Separate record format headings:   *YES
  Record sampling:      *NO
  File review:          *YES   Display all fields in prompt:      *YES

  Take defaults for all selected fields:                          *NO
```

The final type is the *labeled values* presentation display. It appears very similar to a prompt display in that each value is shown preceded by text describing it. All values on this display are output only. By using the Roll keys, the user can see all values not fitting on the initial

Figure 8 Examples of columnar selection displays



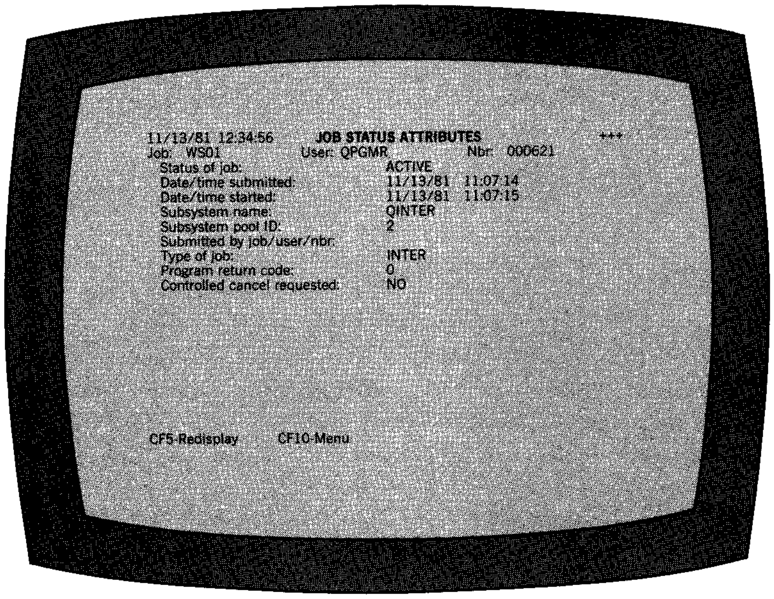
```
11/13/81  12:34:56       CONTROL UNIT STATUS DISPLAY - *ALL
  CTL/DEV      STATUS          JOB NAME     USER       NBR
_ CUD1         ACTIVE
1   RMT1WS01   ACTIVE          RMT1WS01     QPGMR      003012
5 CUD2         ACTIVE
_   RMT21      VARIED ON
_   RMT22      FAILED
_   RMTPRT     ACTIVE/WTR      RMTPRT       QSYS       002983
4 BSCCUD1      VARIED OFF



1-DSPJOB  2-DSP desc  3-CHG desc  4-Vary on  5-Vary off  9-CNLJOB  CF5-Redisplay
```



```
                     QUERY SELECTION LIST
  QUERY        LIBRARY      FILE         DESCRIPTION
_ OPEN         MYLIB        ORDERS       Open order status
_ ORDERS       MYLIB        ORDERS       Order status by account
_ ORDERS5      MYLIB        ORDERS       Order status by company
_ ORDERS6      MYLIB        ORDERS       Order status by salesman



1-Select  2 Query Information  HELP-Help  CF2-Previous display
```

presentation of the display. The title indicates the type of information, and line 2 (and if necessary line 3) shows the identity of the object to which the values pertain. An example is the Job Status Attributes display (Figure 9).

Figure 9 Example of labeled value display



```
11/13/81 12:34:56      JOB STATUS ATTRIBUTES              +++
Job: WS01              User: QPGMR            Nbr: 000621
  Status of job:                    ACTIVE
  Date/time submitted:              11/13/81  11:07:14
  Date/time started:                11/13/81  11:07:15
  Subsystem name:                   QINTER
  Subsystem pool ID:                2
  Submitted by job/user/nbr:
  Type of job:                      INTER
  Program return code:              0
  Controlled cancel requested:      NO





CF5-Redisplay     CF10-Menu
```

In addition to having just four types of displays, five ways are used to optimize function request ease of use within the interactive interface—user menus, menu/prompt front ends, interactive displays, help, and back up/change.

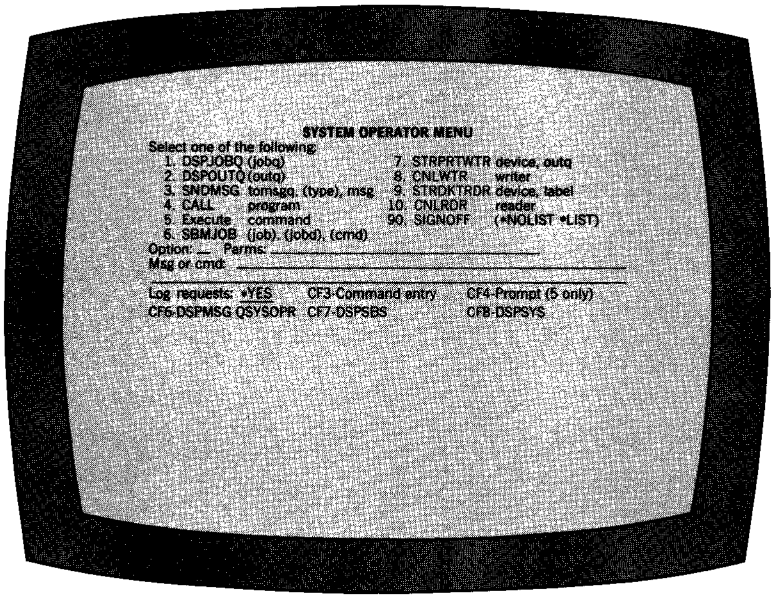
user menus

Among our users there are three primary user types. They are the programmer, system operator, and application end user. By designing a menu for each of these common users, the following is accomplished:

- Greatly improved the system usability for the majority of users
- Provided a way to ease the user onto the system
- Provided an efficient interface for the experienced user
- Provided a model for supplying specialized functional grouping menus for other users

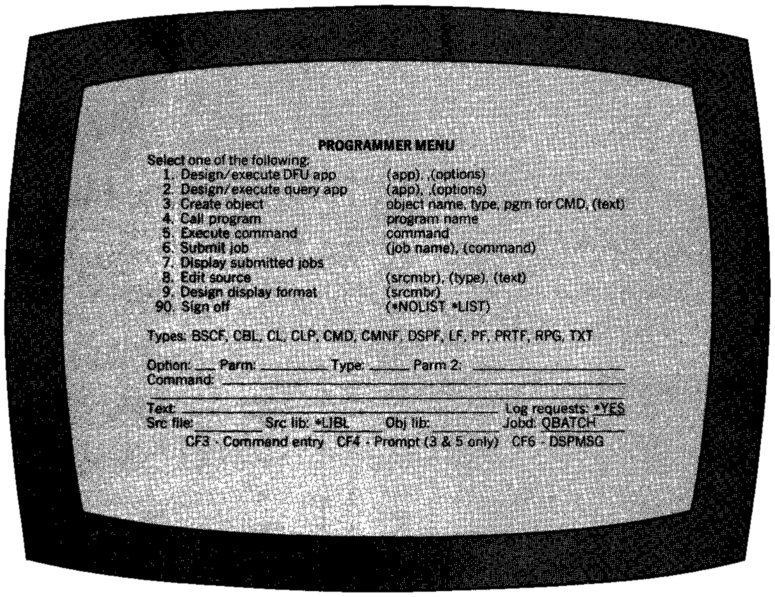
A small number of functions account for the majority of the requests executed by any one user type. A menu is supplied with that small set of high-usage functions on it, thereby relieving the user of the chore of determining what key functions to learn. The users become immediately productive. They only need to key in a number to identify the function they would like to request and the name of the target that they want the function applied to. The necessity of remembering the function and the command syntax required to request it is eliminated.

Figure 10 System Operator Menu



```
                    SYSTEM OPERATOR MENU
Select one of the following:
   1. DSPJOBQ (jobq)                    7. STRPRTWTR device, outq
   2. DSPOUTQ (outq)                    8. CNLWTR     writer
   3. SNDMSG  tomsgq, (type), msg       9. STRDKTRDR device, label
   4. CALL    program                  10. CNLRDR     reader
   5. Execute command                  90. SIGNOFF    (*NOLIST *LIST)
   6. SBMJOB  (job), (jobd), (cmd)
Option: __  Parms: ________________________________
Msg or cmd: ____________________________________________
_________________________________________________________
Log requests: *YES   CF3-Command entry    CF4-Prompt (5 only)
CF6-DSPMSG QSYSOPR   CF7-DSPSBS           CF8-DSPSYS
```

Figure 11 Programmer Menu



```
                       PROGRAMMER MENU
Select one of the following:
   1. Design/execute DFU app        (app), ,(options)
   2. Design/execute query app      (app), ,(options)
   3. Create object                 object name, type, pgm for CMD, (text)
   4. Call program                  program name
   5. Execute command               command
   6. Submit job                    (job name), (command)
   7. Display submitted jobs
   8. Edit source                   (srcmbr), (type), (text)
   9. Design display format         (srcmbr)
  90. Sign off                      (*NOLIST *LIST)

Types: BSCF, CBL, CL, CLP, CMD, CMNF, DSPF, LF, PF, PRTF, RPG, TXT

Option: ___ Parm: _________ Type: _____ Parm 2: _______________
Command: ______________________________________________________
________________________________________________________________
Text: ____________________________________ Log requests: *YES
Src file: _________ Src lib: *LIBL ______ Obj lib: _______ Jobd: QBATCH ____
      CF3 - Command entry  CF4 - Prompt (3 & 5 only)  CF6 - DSPMSG
```

The System Operator Menu (Figure 10) is more command-oriented. The Programmer Menu (Figure 11) is more task-oriented.

Both of these menus are working menus designed to be effective for continuous use by experienced users. Values specified for one option remain to be used for the next related option. For example, in Figure 11, Options 8, 3, and 4 are designed to be used together. By naming the source member the same as the program to be created from it, the program name, type of language source, and a text description can be entered when selecting Option 8 to update the program source. No further specification is needed to create the program and call it, using Options 3 and 4.

Because it is usually necessary to identify the target of a function, the menus are designed with an input area to identify the target and any other required information. Therefore, it is not necessary to go through a separate display to identify frequently entered values. If the needed values are not entered, a message identifies the values that are needed. After the function is complete, the user is back at his/her familiar menu and is ready to perform another operation which might involve the use of that same target. The values remain filled in for the next operation. A set of values only needs to be entered once for a sequence of operations requiring those values. This design makes the menu interface effective for the experienced user who wants to do only a minimum amount of keying for frequently requested operations.

Functions that require more extensive input result in prompt displays with text identifying what input is needed. The text is in terms that a particular type of user will understand.

Menus help the programmer and operator over the hurdle of getting started. Some users prefer to use menus, others start with menus but decide to switch to command entry.[14] System/38 allows this choice. After understanding the basic functions provided on the menu, the user can proceed to more advanced functions. Ways are provided to enter a command on the menu or to request another display or menu to request other functions. Through the use of the base functions, the user sees how the system responds to requests. If the responses are understandable and consistent, the user is left with a feeling of understanding the system and a sense of confidence that might be expressed as "I can even do those functions I have not tried yet."

menu-prompt front ends

A set of menus and interactive prompt displays are provided for sets of function logically used together. The user does not need to use commands for common use functions. These interactive packages of menu, prompt, and presentation displays shield the user at the display workstation from having to enter syntactically structured requests. Instead, the user only needs to select from a list of choices or enter values in labeled input fields. No keywords, parentheses, quotation marks, or commas need be keyed in. Although the displays are nearly self-instructive, help displays are available to answer user questions.

An example of such a set of menu and prompt displays is that provided with the Data File Utility that supports creating, managing, and using data entry applications. Another is the Query Utility that supports creating, managing, and using data base Query report requests. Examples of these Query displays are the second display shown in both Figure 6 and Figure 7. Still another is the Screen Design Aid Utility, which supports creating, managing, and testing display formats for customer applications.

interactive displays

Another ease-of-use approach used on the System/38 is the provision of a convenient means for users to move from interactive display to interactive display or request operations from the display without having to return to the command level of interaction. There are two common types of circumstances in which such movement is done:

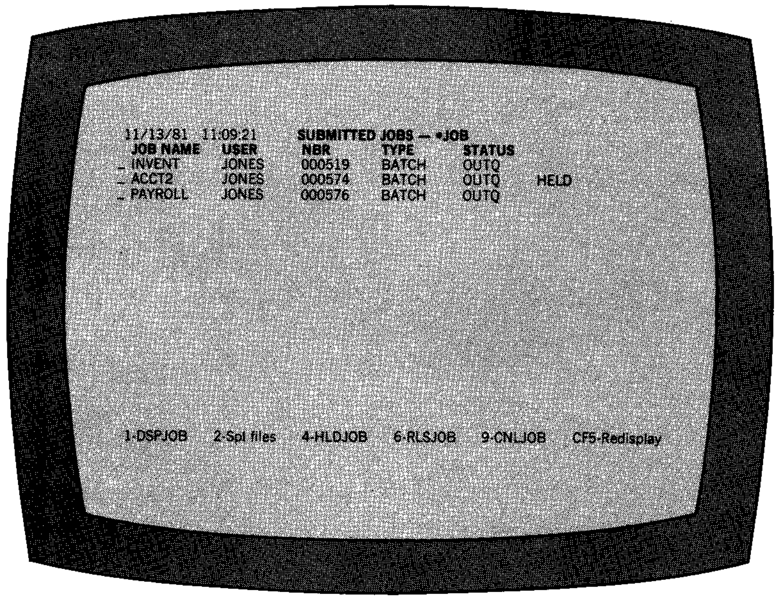
1. If a function allows you to define something, a logical follow-on might be to test or use that function. This situation can exist on the System/38 after a display format is defined through the Design Format function of the Screen Design Aid, through the Design Query Application function of the Query facility, and the design data entry application function of the Data File Utility (DFU).
2. When a function displays the contents, attributes, or status of an object, the user often wants to change it, ask for another piece of information, request another operation against it, or request an operation against one or more other objects identified as part of the information. By allowing such a request directly from that display, the user gains the following benefits:

   - The user does not have to request another display to enter the request.
   - The user does not have to write down the information to remember it until the request can be entered on a subsequent display, thus not only making the task easier but less error-prone.
   - The request requires less keying than if it were entered on a more general display because the user needs only to enter a number next to the item to be operated on or to press a special function key. The number or key identifies the desired operation. No keying is necessary to identify the target because the location of the number or cursor identifies it.
   - If multiple objects are displayed, some key operations may be requested on one or more objects, greatly reducing keying time and saving the multiple interactions involved from having to exit to a command entry display and enter multiple individual requests.

The effectiveness of this approach can be seen by requesting a display of submitted jobs as shown in Figure 12.

From this display the jobs can be canceled, held, or have additional information requested about them. For example, if a two is entered

Figure 12 Example of a display of submitted jobs



```
11/13/81  11:09:21        SUBMITTED JOBS — •JOB
  JOB NAME    USER        NBR        TYPE       STATUS
_ INVENT      JONES       000519     BATCH      OUTQ
_ ACCT2       JONES       000574     BATCH      OUTQ       HELD
_ PAYROLL     JONES       000576     BATCH      OUTQ










1-DSPJOB    2-Spl files    4-HLDJOB    6-RLSJOB    9-CNLJOB    CF5-Redisplay
```

next to the third job, the display of spooled ouput files is shown (Figure 13).

From this display, a user can display the content of one or more files, display the attributes of one or more files, and hold, release, or cancel one or more of the spooled files. Any of these functions can be requested by keying in the option number listed at the bottom of the display. The user is in complete control without having to exit or enter any commands or parameters.

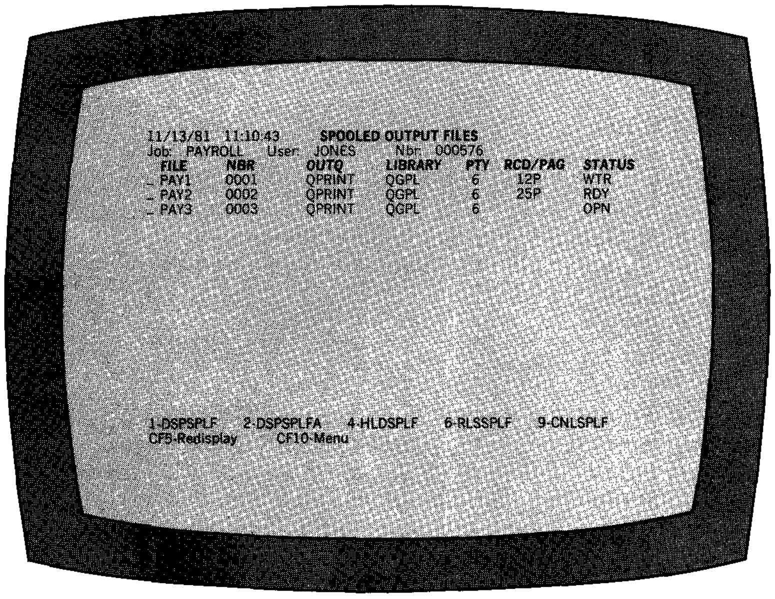
help In end-user functions such as Data Entry, Source Entry, and Query, the HELP key is supported for each display. The resulting help text explains the purpose of the display, what can be done from the display, and how to do it.

Any time an error message is displayed, the cursor can be placed on the message in question, and the HELP key can be pressed to receive an expanded description of the condition and a description of what can be done. It is seldom necessary to consult a message manual.

back up/change The user can normally back up to the previous display. By simply pressing a function key in a sequence of output displays, the user may request to go back and see the previous display again. The user can move back and forth through the sequence as desired.

In a sequence of input displays, the same function key can be pressed to go back and see previously entered values or even change them.

Figure 13 Example of a display of spooled output files



```
11/13/81  11:10:43        SPOOLED OUTPUT FILES
Job: PAYROLL   User: JONES        Nbr: 000576
  FILE      NBR          OUTQ        LIBRARY    PTY  RCD/PAG   STATUS
_ PAY1      0001         QPRINT      QGPL        6     12P     WTR
_ PAY2      0002         QPRINT      QGPL        6     25P     RDY
_ PAY3      0003         QPRINT      QGPL        6             OPN

1-DSPSPLF   2-DSPSPLFA   4-HLDSPLF   6-RLSSPLF   9-CNLSPLF
CF5-Redisplay        CF10-Menu
```

The user can change his/her mind any time prior to a function actually being performed. At any time the user can conveniently change entered values, proceed forward again, go back further, or exit the function.

Concluding remarks

The System/38 user interface was explicitly designed from the outside in to meet the defined usability requirements for the system. System-wide design approaches were adopted and used in the interface design. A strong attempt was made to give it a coherency and consistency that would improve learnability. The coherent design chosen sought to capitalize on both new and proven user interface design principles and approaches.

One major approach was to minimize the number of user interfaces. Three system-wide user interfaces are provided: a single control language for use by the operator and programmer to request externalized system function, a single data description interface to define device and data base data, and an interactive display interface with display formats designed for all three major user types—programmer, operator, and end user. All of these types of users benefit from the integrated design of the System/38 with system-wide user interfaces designed for ease of use.

The programmer benefits because his/her productivity is increased. So that businesses need not add additional staff, we sought to eliminate those tasks that require greater programming expertise. We concentrated effort on each of the areas that require expertise and large portions of a programmer's time and that thus sap his/her productivity. We were convinced that without an all-out effort in the base design, the System/38 would turn out to be just another system with more function and less ease of use.

The programmer in particular benefits from the object orientation because his/her requests correspond directly to what he/she must accomplish. It is not necessary to learn, remember, experiment with, and later maintain a low-level means to indirectly bring about the correct results.

The programmer is freed from having to know the system internals in order to debug programs. System/38 supports an interactive debug capability that allows the user to step through any program, even production programs, to monitor variable values, and to change variables without requiring any hooks to be put into the program. This capability also eliminates the need to predefine where debugging is necessary and the need to compile and recompile to put in, and later remove, the hooks.

The validity checking assists the programmer in two ways. First, it helps to get the work right the first time and eliminate costly repeated attempts. Second, because system validity checking of application display input commands can be used, the programmer is relieved of having to code his/her own for the applications he/she writes.

The single control language allows the programmer to install, configure, operate, test, and define applications through a single set of commands. Any function added by a new release or obtained from other sources can be invoked through a command. All the ease of use of command prompting and validity checking is available to any application by defining a command. Programs can be written directly in CL, or individual CL commands can be executed directly out of any high-level language program. The programmer does not have to learn assembler language to make use of system function.

The system provides a data base where data is defined through a simple forms interface, where like fields only need to be described once, and where interrelationships of fields are defined by using simple references to file names and field names. The high-level language supports references to these already-defined records and automatically extracts the definition of defined variables for each field. This capability makes it possible for the programmer to directly access each field as a variable.

Finally, a complete set of interactive facilities is provided for the programmer, with the Programmer Menu at the center. The programmer can interactively design data entry applications, design query applications to produce reports from the data base, update program source code, design interactive display formats, and submit jobs. Desired information is carried over from option to option so that there is a logical flow during the workstation session. Although the individual functions were not described in this paper, the design approaches that are presented were used in their design.

Feedback from customer surveys, user conferences, and customer visits indicates that System/38 has made significant strides in the area of programmer productivity and ease of use. The high-level consistent interface, centered around the Programmer Menu and the Control Language, has been one of the major contributors to this success.

The system operator has benefited in several ways. Clearly, the single control language, command prompting, and validity checking that provide ease of use for the programmer apply to the system operator as well.

The System Operator Menu gives the operator access to the commonly used operator functions. It allows the operator to conveniently request the interactive selection displays from which he/she can manage the work in the system.

For the end user, a computer is easier to use if he/she sees only the displays and messages that relate to the applications being used. To the extent that "system stuff" intrudes on that application/user relationship, the ease of use for that end user suffers. The System/38 provides several ways for the applications programmer to insulate the end user from the system.

At sign-on, an application program can be called from the user profile associated with the password. The first display the user sees is his/her first application display.

The menus and prompts needed by the end user can be easily provided by the application programmer through Data Description Specifications or the Screen Design Aid menu-prompt facility of the interactive interface. As a result of these definitions, the system provides for validity checking of user input, presentation of messages, appropriate control of keyboard shift, and control of optional device features, such as reverse image and display size, allowing the user to benefit from their use wherever possible.

A menu is provided for end users, called the Program Call Menu. It supports viewing messages, calling a program, sending a message to the system operator, and signing off.

In general, most of the design approaches adopted have been noticed and greatly appreciated by users of the system. The advances in consistency, congruence, and attention to careful interface design have resulted in significant positive user reaction. In many cases the system has raised the level of ease of use that users consider necessary or possible. In other cases users react with "of course, that is the way it should be," or "why don't you improve this area's ease of use too?" As the use of computers increases and more and more users who are not data processing professionals join the ranks of computer users, their expectations and the need for ease of use will continue to rise. This means that current user interface design approaches must be further refined and completely new approaches and technologies must be developed in order to continue the advancement in the usefulness of computers.

ACKNOWLEDGMENTS

The author acknowledges the significant contributions to the System/38 user interface design made by David Peterson, the software human factors engineer assigned to the System/38. He worked with the author throughout the primary design period. Mr. Peterson also served as a key contributor and reviewer for this paper.

The author also recognizes the component team leaders for their concern for usability in the design of their portions of the system and the members of the Design Control Department for their cross component attention to usability and coherence of the system.

Special thanks are due to David Harvey, Design Control Department manager, for his continued support for usability, for his standard of excellence for this as well as the other key factors in the design of the system software, and for serving as a key reviewer for the paper.

CITED REFERENCES

1. J. Crane, "Trends in DP budgets," *Datamation* **27,** No. 5, 140 (May 1981).
2. L. Runyan and W. Schatz, "Application development," *Datamation* **27,** No. 3, 165 (March 1981).
3. D. E. Peterson and J. H. Botterill, "System/38—An IBM usability experience," *Proceedings—Human Factors in Computer Systems,* Conference, Gaithersburg, MD (March 15–17, 1982).
4. K. W. Pinnow, J. G. Ranweiler, and J. F. Miller, "System/38 object-oriented architecture," *IBM System/38 Technical Developments,* 55–58, G580-0237, IBM Corporation; available through IBM branch offices.
5. J. C. Thomas and J. M. Carroll, "Human factors in communication," *IBM Systems Journal* **20,** No. 2, 237–263 (1981).
6. D. G. Harvey and A. J. Conway, "Introduction to the System/38 Control Program Facility," *IBM System/38 Technical Developments,* 74–77, G580-0237, IBM Corporation; available through IBM branch offices.
7. B. Shneiderman, *Software Psychology,* Winthrop Publishers, Inc., Cambridge, MA (1979).
8. A. Reed, "Error-correcting strategies and human interaction with computer systems," *Proceedings—Human Factors in Computer Systems,* Conference, Gaithersburg, MD (March 15–17, 1982).
9. J. H. Botterill and W. O. Evans, "The rule-driven Control Language in System/38," *IBM System/38 Technical Developments,* 83–86, G580-0237, IBM Corporation; available through IBM branch offices.

10. G. W. Furnas, L. M. Gomez, T. K. Landauer, and S. T. Dumais, "Statistical semantics: How can a computer use what people name things to guess what things people mean when they name things?," *Proceedings—Human Factors in Computer Systems*, Conference, Gaithersburg, MD (March 15–17, 1982).
11. J. Martin, *Design of Man-Computer Dialogues*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1973).
12. S. E. Engel and R. E. Granda, *Guidelines for Man/Display Interfaces*, Technical Report TR 00.2720, IBM Corporation, Poughkeepsie, NY (1975). (ITIRC No. 76A 00235).
13. D. E. Peterson, "Screen design guidelines," *Small Systems World* **6**, No. 8, 19 (February 1979).
14. D. Gilfoil, "Warming up to computers: A study of cognitive and affective interaction over time," *Proceedings—Human Factors in Computer Systems*, Conference, Gaithersburg, MD (March 15–17, 1982).

GENERAL REFERENCES

The following publications from the IBM Corporation are available through IBM branch offices.

IBM System/38 Control Language Reference Manual, SC21-7731.

IBM System/38 Control Program Facility Concepts Manual, GC21-7729.

IBM System/38 Control Program Facility Programmer's Guide, SC21-7730.

IBM System/38 Control Program Facility Reference Manual—Data Description Specifications, SC21-78606.

IBM System/38 Data File Utility Reference Manual and User's Guide, SC21-7714.

IBM System/38 System Introduction, GC21-7728.

IBM System/38 Query Utility Reference Manual and User's Guide, SC21-7724.

IBM System/38 Screen Design Aid Reference Manual and User's Guide, SC21-7755.

IBM System/38 Source Entry Utility Reference Manual and User's Guide, SC21-7722.

IBM System/38 Technical Developments, G580-0237.

The author is located at the IBM System Products Division headquarters, 44 S. Broadway, White Plains, NY 10601.

Reprint Order No. G321-5174.