1984

The Gshell: a graphical command language for UNIX operating system

Andrew Kitchen

Rochester Institute of Technology

School of Computer Science and Technology

THE GSHELL

A Graphical Command Language for

the UNIX Operating System.

by

Andrew Kitchen

Approved by: Signature not legible

Signature not legible

Signature not legible

itle of Thesis: The Gshell: A Graphical Command Language for the Unix Operating System

_____ hereby (grant/deny) permis-
ion to the Wallace Memorial Library, of RIT, to reproduce my thesis in
hole or in part. Any reproduction will not be for commercial use or profit.

or

ANDREW KITCHEN prefer to be contacted each
ime a request for reproduction is made. I can be reaced at the following
ddress:

School of Comp. Sci. & Tech.
Rochester Inst. of Tech.
X6540

)ate: Feb 10, 1984

ABSTRACT.

The Gshell, a graphical command interpreter for Unix, and an executing and editing environment for the Gshell system are described. Through the use of a pictorial representation of command scripts the Gshell system provides a more powerful expression of concurrency than is possible with conventional Unix command languages. Its syntax and use is explained and major features of its underlying control and data structures are discussed from the point of view of the designer and the programmer.

Keywords: Unix, C, shell, command interpreter, graphical language, concurrency, process, pipe.

TABLE OF CONTENTS.

I. THE PROBLEM.

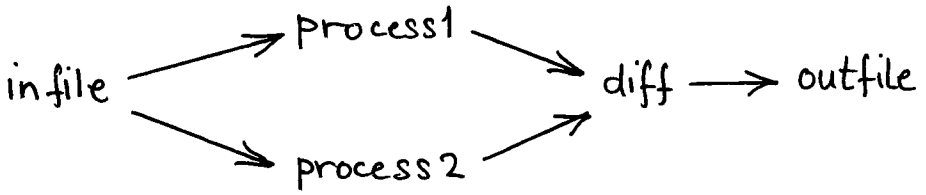
1. Problem Statement.

The standard shells available to the Unix user provide process communication by means of pipes, [9], [3], [7], e.g.

```
sort <file | pr | lpr
```

However, the use of pipes is limited to filters, that is, processes with a single input and single output. D. M. Ritchie, in [11], discusses more extensive pipelining facilities. He mentions that, in the design of the user interface for Unix, the provision of a more powerful pipelining structure was considered. The essentially linear nature of the standard shell syntax together with the potential for deadlock in a general network of communicating processes were considered major deterrents against the development of more general facilities. For these reasons the idea was dropped.

The linear nature of standard computer input is a problem faced by any designer of a language intended to represent parallelism. An operation such as



(where diff is the Unix command for comparing two files) is most clearly represented by a combination of text and graphical notation. In fact flowcharts and data-flow graphs, [5], provide a natural notation on which to base a representation of parallel processing. A limitation of such an approach, however, is the traditional lack of appropriate I/O devices for

graphical output. The wealth of GIGI graphics terminals at R.I.T. makes it a suitable environment in which to experiment with a graphical command language.

The problem of deadlock is one I did not address. I am hopeful that the design of the Gshell language is simple and clean enough that this problem will be mitigated and that, in any case, some experience with the problem may be gained by the development and use of such a system.

The design of the Gshell is influenced by the structure of functional languages, such as LISP, [6], and by the recently developed data-flow languages (e.g. LUCID, [1], and VAL, [10]), particularly in its variable-free notation. However, a major influence on its structure and appearance comes from message driven parallel processing systems, such as ITP, [13], and Smalltalk, [15]. In particular, the design of the programming environment is modeled somewhat loosely on the Smalltalk user interface, [12] and [2]. The Gshell system is intended to provide an almost "modeless" environment, that is, one in which the user can slip from one task to another, and back again, without compromising the integrity of the tasks that have been left incomplete. Furthermore, it is intended that programming in the Gshell should be done with a minimum of typing.

2. Previous Work.

As indicated in the preceding section, a multi-pipelining interface to the Unix operating system was considered by its designers, it was never developed, [11]. Other graphical programming languages do exist, from the discredited flowchart to formal data-flow program graphs, [8] and [4], being designed currently in anticipation of the next generation of supercomputers. There are other graphical languages, such as AMBIT/L, [14],

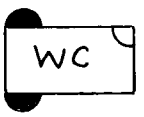
designed as a graphical representation of LISP. Smalltalk provides a user environment of sophisticated design with features which minimize the linear nature of the man/machine interface. However, programs in Smalltalk consist of text, organized into lines, with no provision for the explicit representation of parallelism.

II. THE GSHELL USER INTERFACE.

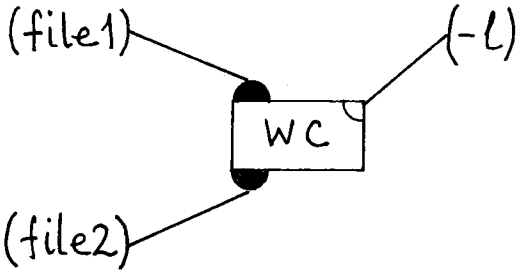
In this chapter the Gshell is described from the user's perspective. In the first section the graphical tokens of the language are introduced and the syntactical and semantical structure of Gshell tableaux (analogues of the command scripts of other Unix shells) are discussed. The second section presents the environment within which tableaux are created and executed. The chapter ends with a description of the commands provided for the creation, editing, and execution of tableaux.

1. The Gshell Language.

A traditional shell command consists of a command name followed by a list of arguments, each of which is either a file name or a flag. Furthermore, many commands assume the existence of default input and output "ports", standard input and standard output, directed to the user's terminal but capable of being redirected to files or to the I/O "ports" of other commands via "pipes". In the Gshell, a command is represented by a labeled box, called a command or process box, with tabs to represent input and output ports. For example:



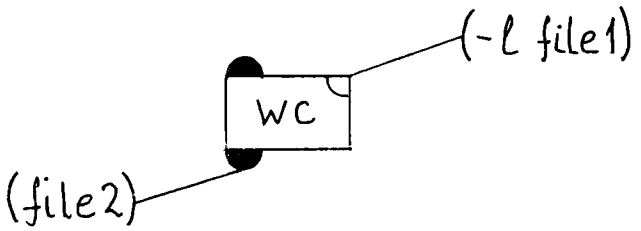
which can be used as follows:



This corresponds to the more familiar

wc -l <file1 >file2

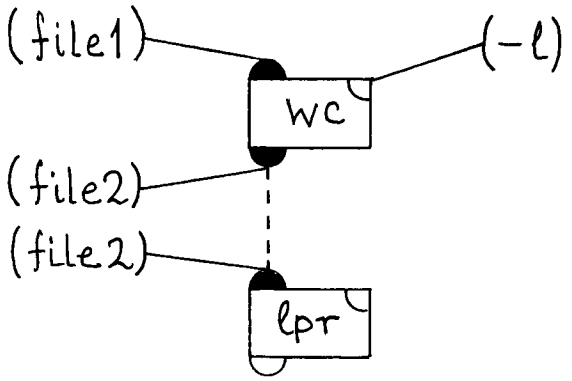
Note: wc -l file1 >file2 is subtly different:



Input is through the tabs (only one in the example shown here) at the top and output from those at the bottom of a command box. An unlabeled input (resp. output) tab takes its input from (resp. delivers its output to) the terminal. Thus the input and output tabs represent a generalization of standard input and standard output. A tab, the flag tab, is provided for those parameters, such as flags, which would normally be part of the command line of a shell command. An unlabeled flag tab indicates that no flag is present.

Sequential sychronization of processes is indicated by the channel symbol

For example,



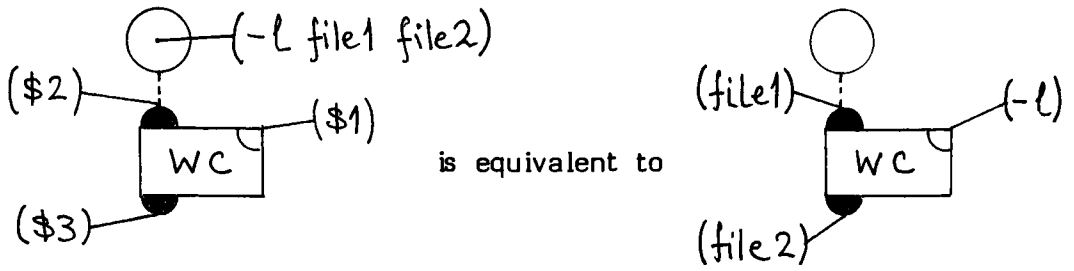
corresponds to

wc -l <file1 >file2; lpr file2

The unshaded output tab on the command box for lpr is a sync-tab, used purely for synchronization of command boxes that do not deliver redirectable input or output. (Note: the term redirectable I/O refers to a flow of data in or out of a Unix or Gshell command that can be directed to and from files, devices, and other commands. Redirectable I/O is a generalization of the standard input and output of Unix and C.) Sync-tabs may have at most one input or output, namely, a single channel.

A group of boxes and parameters linked together, as shown in the diagram above, is referred to as a tableau. Tableaux correspond to the command scripts of other Unix shells. They can be viewed, roughly speaking, as programs in which the command boxes play the parts of statements or procedure calls; though it should be understood that this view ignores the issue of concurrency.

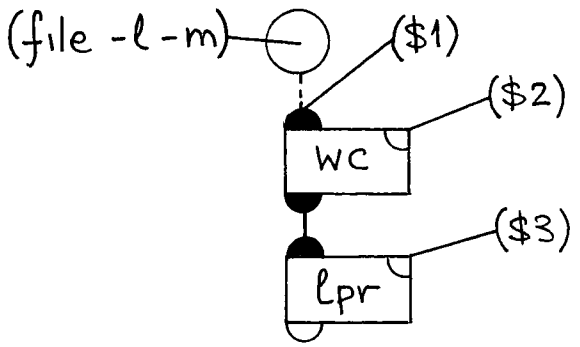
Argument lists may be introduced into a tableau, and individual arguments selected, by means of positional parameters similar to those used in other shells, e.g.,



By convention, all tableaux begin with the start symbol

The argument list introduced into a tableau through the start symbol is made available to every node in the tableau simultaneously.

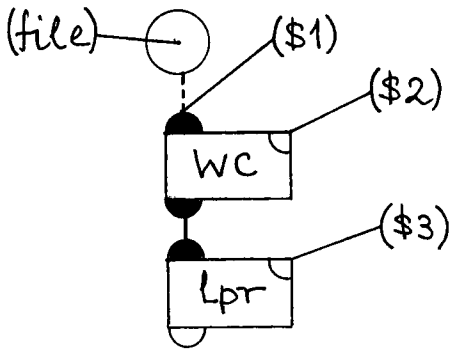
Unix pipes are represented by solid lines. Thus, this tableau



is equivalent to the following Cshell script

```
wc -l <file | lpr -m
```

Positional parameters are ignored if the corresponding term in the input list is missing, e.g.

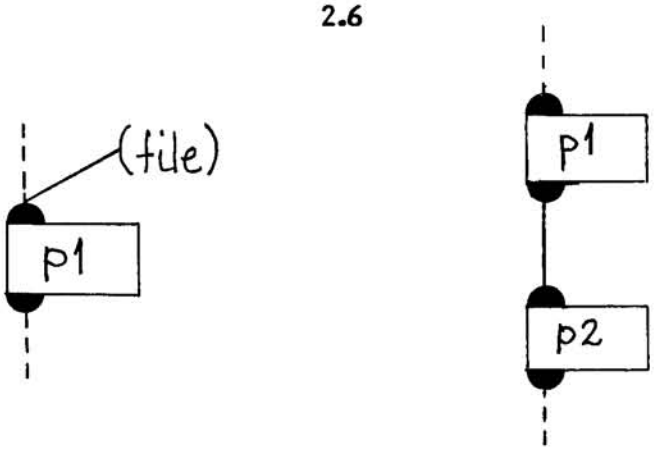


is equivalent to

wc <file | lpr

A tab may be the source or destination of at most one pipe and one channel. In fact, an input tab (resp. output tab) that is not a synch-tab provides at most one input (resp. output) data stream and one synchronization channel. A synch-tab provides synchronization alone. Thus, for example, an output tab may either be connected by a channel to an input tab on another command box while simultaneously directing output to a file or may direct its output to the same destination as the channel.

To summarize the descriptions given above: a link which provides synchronization between command boxes is denoted by a broken line and is called a channel. If data other than synchronization information is passed then the link is drawn as a solid line. Notice that data can pass between a parameter, say a file, and a command as well as between two commands. In the latter case the data and synchronization links have the same destination and the pair represents a true Unix pipe. In discussing the Gshell, we use the term pipe to indicate any link along which data passes, for example, the link between a file and a command which reads from that file. The following diagrams summarize the typical uses of pipes.



It should be noted that pipes and channels provide the only means of synchronization and communication between the processes associated with the command boxes of a tableau. The execution of individual nodes is initiated by the receipt of the exit status of preceding nodes via the channels:

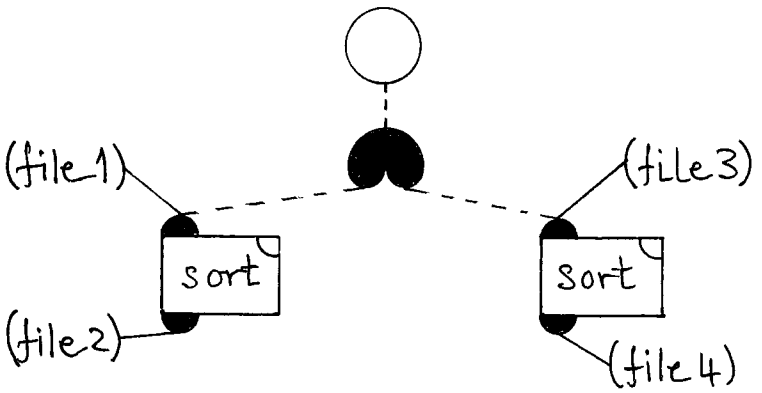
When p1 is done, the exit status of p1 is passed to p2 which begins. When it is complete, its exit status is passed to p3.

When p1 completes, exit status of p1 is passed to p2 which begins and passes the exit status of p1 on to p3, which also begins and uses the output of p2 as it is produced.

The execution of a tableau is not complete until the execution of every node has terminated.

A special object, called a tee

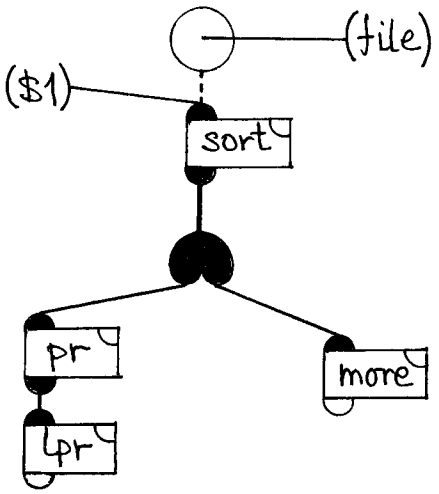
is provided to split a data or control link. Thus:



is similar to

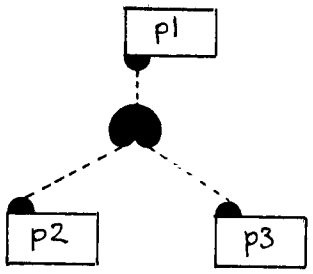
```
sort <file1 >file2 & sort <file3 >file4
```

that is, the two sorts proceed in parallel. Similarly, the tableau

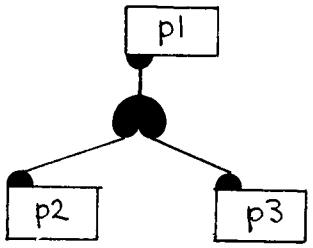


prints a copy of the sorted file on the line printer and displays it at the terminal.

The following examples illustrate the synchronization of command processes as this applies to the tee:

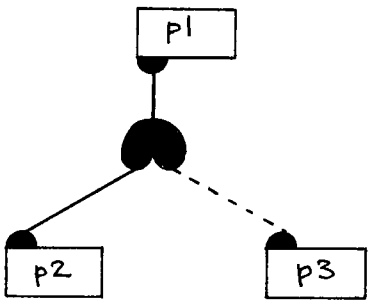


p2 and p3 do not commence until p1 signals that it is done. They, then, execute concurrently.



In this case, p2 and p3 execute in relative synchrony with p1, simultaneously absorbing the output of p1 as it is produced.

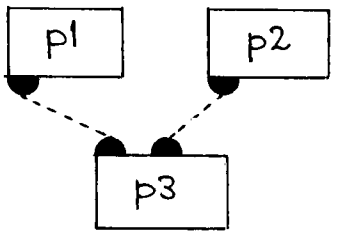
Other combinations are also possible, e.g.,

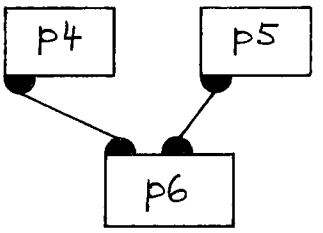


Here, p1 and p2 execute concurrently with p2 reading the output of p1. When p1 exits, p3 commences execution. Thus, p3 cannot start until p1 has finished but the execution times of p2 and p3 may overlap.

The tee provides a mechanism for splitting, or duplicating, the flow of control and data, so that different sections of a tableau may run concurrently without being in direct communication with one another. The Gshell does not provide any kind of "join" node to combine two control or data flows, as there is, clearly, no universal way to perform such a function. However, process boxes may have more than one input (and/or output) tab:

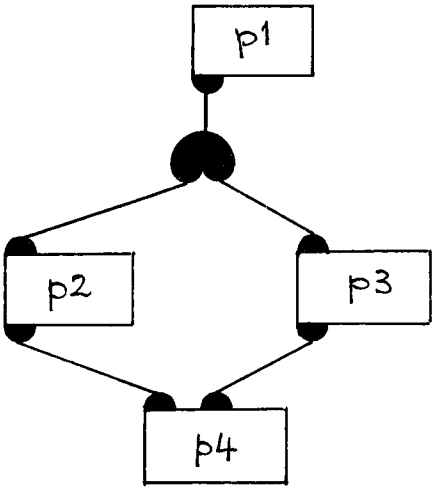
Thus, the following are legal configurations





p3 must wait for both p1 and p2 to complete. On the other hand, p6 absorbs input simultaneously from p4 and p5. The joining of the two data flows from p4 and p5 is controlled by the process p6, and so is not handled directly by the Gshell.

Note: A tableau having the potential for deadlock results from configurations of the following form:



For example, in this case, if p2 produced output faster than p4 was able to consume it, the buffer maintained by the Unix system for the pipe between these two processes would eventually fill. At this point, p2 would wait for p4

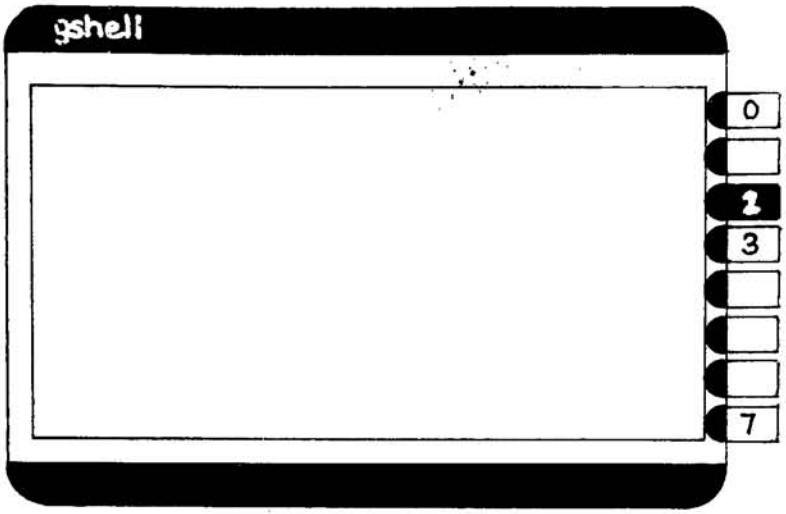
to clear the buffer. If p2 were blocked for long enough, p1 would eventually halt, too, cutting off the flow to p3 as well. In a like manner, therefore, p3 would stop. There would not necessarily be any danger in this, unless the reason that p4 was so slow in reading the pipe from p2 was that it was waiting for input from p3. However, if that should be the case, the tableau would be deadlocked.

As was mentioned in the first chapter, the design of the Gshell does not include any mechanism for avoiding such an eventuality.

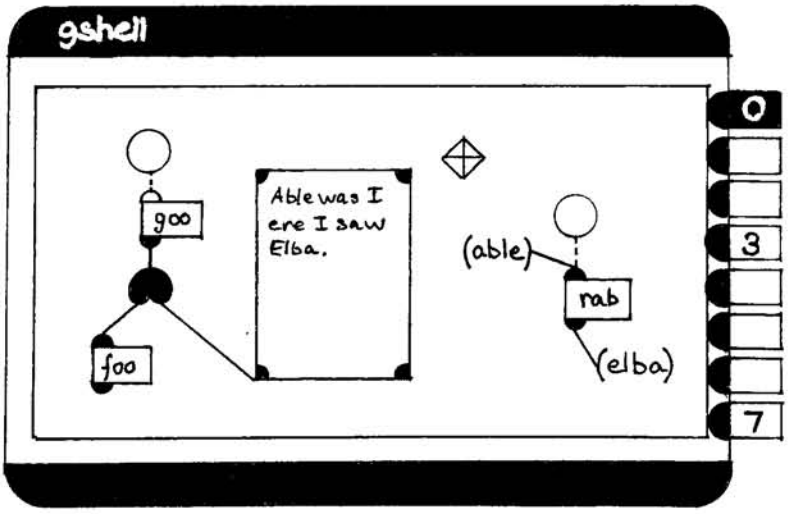
2. The Gshell Environment.

The environment within which a user of the Gshell works consists of a set of eight pages. The pages are arranged in a pile, only the top of which (the top page) is visible to the user at a given time. All pages have reference tabs on their right hand sides, by which they can be addressed and called to the top of the pile at the user's convenience. The system assigns each page a single digit number. If a page is not empty, then its number is displayed on the associated page tab. The tabs of pages which are empty remain blank. The tab associated with the page the user is currently working on is highlighted in reverse video.

See diagram on next page



Each page consists of two fields and an identification tab. Across the bottom lies a one line field, the command/message field, displayed in reverse video and showing the most recent command or action taken. Error messages are also displayed here. Above this field lies the workspace. This comprises most of the page and may contain one or more tableaux and/or data panels (unnamed subpages containing input for or output from tableaux).



As the environment of the Gshell is richer than that of other shells, a set of meta-commands is provided for its manipulation. These we will refer to as Gshell Editing commands. They will be discussed in detail in the next section.

3. Gshell Editing Commands.

Gshell commands can be divided into the following categories.

(1) Cursor Control.

(2) Movement and Scaling.

(3) Deletion and Insertion.

(4) Execution.

In this section we will discuss the syntax and semantics of the Gshell commands. Formal syntax diagrams for these commands are listed in the Appendix.

Requests for the operations listed above are communicated to the Gshell as special keystroke sequences. The mnemonics used in the descriptons which follow indicate fixed sequences of ASCII characters. If a programmable keypad or special function keys are available (as on the DEC GIGI or VT125 terminals) then each boxed character string (e.g., INS or |) can be represented by a single keystroke. The following diagram shows the GIGI keypad as it should be labeled for use with the Gshell.

<table>
<tr><td>INS</td><td>MOV</td><td>WIND</td><td>DEL</td></tr>
<tr><td>REF</td><td>SEL</td><td>PAGE</td><td>START</td></tr>
<tr><td>TEE</td><td>CMD</td><td>DPAN</td><td>PARAM</td></tr>
<tr><td>PIPE</td><td>CHAN</td><td>IF</td><td rowspan="2">EXEC</td></tr>
<tr><td colspan="2">END</td><td>FOR</td></tr>
</table>

Cursor Control.

As described in the preceding section, the Gshell environment consists of a number of pages, each containing objects (command boxes, tees, parameters, etc.) linked together to form tableaux. The user must be able to create objects at any position on the screen and later rearrange them at will. These positioning functions are provided by the cursor keys, [↑], [↓], [←], [→]. The cursor keys are also used to adjust the size of a data panel, once it is placed on a page, and they play a part in the insertion of links between objects. As the cursor keys control a number of positioning and scaling functions the representation of their operation on the screen changes to reflect their changing roles. This representation may take the form of a cursor (e.g., ⊕ when selecting object position, + for object movement, and ⌟ for data panel scaling) or a flashing line (when inserting linkages).

When control is at the Gshell Editing command level (represented by the ⊕ cursor) the user may move the cursor to any position on the screen to select an object currently displayed or to select a location for the insertion of a new object. Once an object or position has been selected, the user must press other keys to command the Gshell to perform the actions desired. Once a legal key has been pressed the command level cursor disappears, only to reappear when the requested operation has been completed. Depending on the command invoked, other cursors or indicators may appear on the screen to guide the user. These will disappear when control returns to the command level.

Movement and Scaling.

An object currently displayed can be moved to any other position on the screen. To do this, the command cursor is positioned so that it lies on

top of the object, then the [MOV] key is pressed. This causes the command cursor to be replaced by the move cursor +, This, too, is controlled by the cursor keys. To exit from the "move" mode, the user presses the [END] key. The object selected is then deleted from its current position and redrawn at the final position of the move cursor. At the same time the move cursor is erased and replaced by the command cursor. The above sequence is a paradigm for most of the Gshell commands. Namely, press a command key, perform a sequence of operations appropriate for the mode selected then, finally, press [END] to return to the command level.

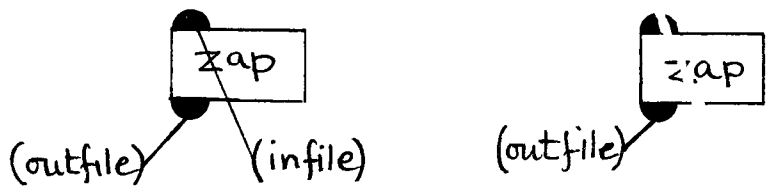
Another command, almost identical to the move command in syntax, is the window command, initiated by the [WIND] key. This applies only to data panels. It allows the user to adjust the size of a data panel by means of the cursor keys. Again, exit from the "window" mode is effected by pressing [END], at which time the data panel is drawn to the dimensions selected by the user.

Finally, there is another command which could be construed as a kind of movement. A new page can be selected by positioning the command cursor over the node of the required page and pressing the [PAGE] key. The current page is then erased and the contents of the new page are displayed on the screen.

Deletion and Insertion.

Of course, objects have to make their appearance on the screen somehow and later, when they have served their purpose, the user must be able to remove them. This is not the whole story, however; links between objects must also be inserted and removed. The two-level nature of insertion and deletion leads to a more complex command structure.

The insertion and deletion of objects follows the pattern established above. In fact, object deletion is the simplest operation of all, namely: move the command cursor over the object, then press the DEL key. A command of similar syntax, which is often used in conjuction with DEL, is the refresh command, REF, which redraws the selected object. This is useful for repairing object images in the situation where overlapping objects have gotten partially erased by delete or move operations:



To insert an object, the user first selects a location on the screen free from other objects, then presses the insert key, INS, followed by a second key to indicate which object type is to be inserted. Some objects, e.g., tee, data panel, and start, require no further description. Such an object will be drawn immediately, after which control will return to the command level. Other objects are "nameable", e.g., commands and parameters. These require the insertion of text. This requirement is signalled by the appearance of an arrow cursor, ↗, at the selected position on the screen. The user then types in the required text, which is echoed in the command field, followed by the END key.

Text for a parameter node is accepted without comment. If, however, the object the user is attempting to insert is a Unix command, the system will look up the command name in a table to find out how to draw its node. If the name is not in this list, then the insertion is rejected. The installa-

tion of Unix commands in this table is a simple task, as each entry consists of just two character strings: the command name and a string of single character codes indicating the tabs associated with the command node.

The insertion and deletion of links requires a means of selecting a tab or link on which to perform the operation. This is done with the select key, [SEL] . First, the user moves the command cursor over the object whose links are to be modified. A single stroke of the [SEL] key selects the first tab on the object, the top left-hand one. Repeated striking of this key causes selection to cycle through the object's tabs, left to right, top to bottom, and then back to the top again. A selected tab is highlighted by causing it to flash. If a link is attached to the tab, it, too, will flash. Some tabs may have both a control link (channel) and a data link (pipe) attached to them. In such a case, the two links are selected separately; first the control link, then the data link.

To delete a link, the link is first selected then the [DEL] key is pressed. After the link has been deleted, the tab and the remaining link to that tab (if there is one) will be highlighted. The system is still in select mode. Other nodes may be selected by striking the [SEL] key again. Return to command level can be effected by pressing [END] .

Link insertion is the most complicated operation syntactically, entailing, as it does, the nesting of two modes: insertion within selection. First, a tab is selected on a given object, then the [INS] key is pressed. A line, the cursor line, can then be extended from this tab by means of the cursor keys. To make a link to another object it is necessary to attach the free end of this cursor line to a tab on that object. The object is chosen by moving the end of the cursor line over it. Then, a particular tab is selected by repeat-

edly pressing the [SEL] key. For the actual insertion of the link the user must press either [CHAN], for a control link, or [PIPE], for a data link. The striking of either of these keys also causes an immediate exit from "insert" mode and a return to "select" mode. Furthermore, at any time while in insert mode the user may exit from it by pressing [END].

3.4. Execution.

Once a tableau has been built, execution of it is a simple matter. The command cursor is placed over the start node (every tableau must have at exactly one start node) and the [EXEC] key is pressed. If keyboard input is required, this can be typed in and will be echoed in the command field. The end of input is signalled by typing Control-C (CTRL-C). The system indicates that execution is complete by redisplaying the command cursor in the start node of the tableau just executed.

III. DESIGN DETAILS OF THE GSHELL.

1. The Gshell Data Structures.

The contents of each of the Gshell pages are stored in a file: page.0, page.1, etc.. When a page is selected to be displayed, the contents of its file are read into memory. There are two major data structures associated with the currently displayed page (currpage).

First, associated with each object to be displayed is a record describing the object's type, its size, location, the number and nature of its tabs, and the destinations of any links attached to it. This we will refer to as an object node. In fact, object and node will be used interchangeably for the graphical representation of the object and for the underlying record that describes it. The formal node structure is as follows:

```
struct onodetype {
        int ord;
        int oclass;
        int status;
        int drawflag;
        char name[NAMEMAX];
        int x,y;
        int wr,h;
        struct {
            char tclass;
            int filenum;
            struct onodetype *clink;
            int ctab;
            struct onodetype *dlink;
            int dtab;
        } tab[TABMAX];
}
```

The descriptions of these fields are as follows:

ord

An ordinal number associated with the node. It is used primarily for numbering nodes prior to storing them in a page file, so that pointer

values associated with tab links can be translated into a representation that is independent of the actual memory allocation for individual object nodes.

oclass

This indicates the type of object that is represented by this node: tee, command, etc.

status

During execution of a tableau, this field indicates whether the execution of the associated object is yet to be initiated, is in progress, or is complete. It is also used, in the page nodes, to indicate whether the associated page contains text or tableaux.

drawflag

This is used, during the drawing of a page, to mark those objects that have already been drawn.

name

For a command or parameter node, this field stores the character string entered by the user at the time the node was created. For each nonempty data panel this field contains the name of a file which holds the text displayed in the data panel. The form of this file name is as follows:

page.n.xxx.yyy

where n is the number of the page on which the panel lies and (xxx,yyy) are the coordinates of the position of the panel on the page (the location of its upper left-hand corner).

x,y

These two values are the coordinates of the object's position on the page.

wr,h

Some objects (commands, data panels, etc.) are roughly rectangular and so are represented by a width and a height. Others (tees and start nodes) are roughly circular. These are represented by a radius. In the latter case the field wr contains the radius and h is ignored.

Associated with each node is an array of potential tabs. Currently, a maximum of 6 (TABMAX=6) is supported. Each tab may have at most 2 links attached, a control link and a data link. Each linkage is completely defined by specifying the object which is its destination and the number of the tab, on that object, to which it is attached. The object specification is handled by providing a pointer to the node which contains the object's record. From this perspective the tab structure is quite straightforward.

tclass

This describes the nature of the tab; whether it is an INTAB, OUTTAB, FLAGTAB, etc.

filenum

This field is used, during execution of the node, to hold the file descriptor associated with the data link to this tab. This file descriptor is associated with either a file or the "read" or "write" end of a pipe.

clink, ctab, dlink, dtab

These fields describe the control and data links.

All the information required to draw and execute an object is stored in its associated node. However, in order to insert, move, or delete the object, the user, and thus the system, must have some way of associating it with a location on the screen. This identification is handled by means of the second major data structure, the page map. This is a rectangular array of linked lists of pointers to objects in the current page. It is described formally as follows:

```
struct pnodetype {
        struct onodetype *obj;
        struct pnodetype *next;
} *pagemap[YGRIDMAX][XGRIDMAX];
```

Currently XGRIDMAX=16 and YGRIDMAX=10. The workspace of the current page is divided into a grid of 160 (XGRIDMAX*YGRIDMAX) squares. With each of these squares is associated a component of the page map array. This component is the head of a linked list of pointers to those objects on the page that overlap this grid square (see figure 1.).

When the user issues a command that requires the selection of an object, the system calculates which grid square the cursor currently lies in. It then traverses the linked list associated with that square (pointed to by the page map), checking the cursor location against the exact location and dimensions of each object in the list to determine whether the cursor lies over the object. The linked list is always traversed completely, as the system must know whether the selection is well defined or not. For example, the user might attempt to move an object while actually pointing to two overlapping objects.

Note that the purpose of the page map is to reduce the search time for the node of an object selected by the user. In the present implementa-

tion each grid square is small enough that it normally intersects no more than one object, so that the required object is usually the first one encountered. The search could be performed more simply if the nodes of all the objects on the page were held in one linked list, but this process would take longer.

These two data structures, the object nodes and the page map, are powerful enough to handle most of the operations associated with the editing and execution of tableaux. However, two other structures should be mentioned briefly to round out the description of this aspect of the Gshell.

When the user selects a new page, the old page must be copied into its file (page.0, page.1, etc.) and the objects must be loaded from the new page file into memory, with pointers to their nodes inserted into the page map. The major difficulty with this process involves the recording of the linkage structure (between objects) in the page file. The actual pointers cannot be copied into the file, as they will bear no relation to the new pointers allocated when the page is selected again. Therefore, the object nodes are numbered consecutively (using the ord field) and, as the page is copied to the disk, each linkage is replaced by the ordinal number of its destination. The assignment of numbers to the object nodes proceeds as follows: all pointers to each object, in turn, are removed from the map and a pointer to that object is stored in a simple array of object nodes, onodelist. When all objects have been added to this list, the index of each object pointer in the list becomes the object's ordinal number. Finally, the object nodes are copied from this list to the page file, with each link pointer replaced by the ordinal number of its destination.

The last data structure makes its appearance during execution. The system must keep a record of all commands which are currently running, associating process numbers with object pointers. In this way, when a process exits the command node associated with it can be identified and any successor commands initiated. This information is stored in a linked list whose structure is self-explanatory:

```
struct procnodetype {    /* node in process list */
   int process;          /* process number       */
   struct onodetype;     /* pointer to object    */
   struct procnodetype; /* to next process node */
}
```

2. The Execution of Gshell Tableaux.

Though the Gshell is , of necessity, relatively complex, the design of the procedures which handle the editing of tableaux (moving, selection, insertion, and deletion) is a straightforward exercise in structured system design. Routines for adding and removing objects from the page map must be written: these provide the principal mechanisms for moving, inserting, and deleting objects as well as for changing from one page to another. The process of object selection was discussed in the chapter on Gshell data structures. The selection of individual tabs and links, on the other hand, requires a cycling through the list of tabs associated with the object. This is made more complicated by the requirement that the links to each tab be cycled through, as the sequence of links to the tabs of an object will change as links are inserted and deleted.

A major challenge in the implementation of the system comes in the modularization of the description of objects so that they can be drawn and erased efficiently, and so that individual tabs and their associated links can

be highlighted when selected. These, though challenging problems, are similar to ones met by a software engineer in the design of any graphical system. Of more interest is the design of those modules that handle the execution of a tableau once it has been created.

Looking at a tableau as a static object, the synchronization and communication structure of the group of processes that define its execution is described by the links (channels and pipes) between its nodes. For the discussion that follows it is helpful to imagine that only start, tee, and command nodes actually have processes associated with them and that other nodes (pages, data panels, and parameters) are just sources or destinations for the data flow. In practice a separate process must also be associated with every object which might modify the screen at run time (for example, data panels that are being written to). However, from a theoretical point of view, this is an implementation issue.

Nodes connected solely by control links must be processed sequentially. Nodes connected by control and data links, i.e. pipes, must execute concurrently and must have the means to transmit and receive data. These two conditions require that the system keep track of the birth and death of every process. When the Gshell system initiates the process for a node, it must also initiate processes for all nodes connected to the given one by Gshell pipes. Furthermore, it must establish an actual Unix pipe for each of the Gshell pipes. On the other hand, when a process dies its successors (the processes of nodes linked to its out-tabs by channels) must be initiated. Strictly speaking, the correct way to do this is to decompose the whole tableau by dividing it up into subgroups of nodes whose members are connected by pipes but not channels, and then execute each subgroup as a unit,

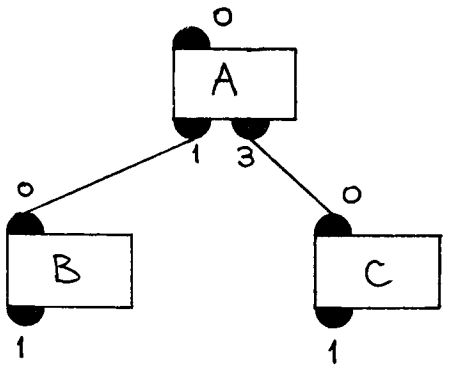
initiating it only when all its predecessors have completed execution. A simpler approach, the one taken in the current design, is to execute the tableau node by node beginning with the start node and observing the conventions on birth and death described in the paragraph above. This approach appears to be adequate. However, it may not be as robust in the face of a large flow of data between nodes, as there could be an appreciable delay between the initiation times of processes which are only indirectly connected by pipes.

The difficulties which arise in the design of software involving multiprocessing are likely to come as much from the peculiarities of the concurrency and communication primitives provided by the host operating system as from any inherent complexity in the design. This is certainly true in the designing of the Gshell to run under Unix. The ease with which programs can interact with the underlying operating system through such functions as execl and fork is balanced by the lack of selectivity provided by wait and the fussiness of dealing with the file descriptors returned by open, creat (sic), and pipe using dup. In this regard, a vote of thanks is due the person who designed the function dup2. Without it, the housekeeping problems involved in the assignment of validly numbered file descriptors to the pipes and files associated with different processes would have been intolerable.

An expansion of these cryptic remarks is in order here, particularly as they refer to one of the trickier parts of the Gshell design. For the Gshell to be truly flexible -- that is, a true two-dimensional analogue of the more traditional Unix shells -- the concept of standard input and standard output has to be extended to the case where a Unix command has more than one redirectable input and/or output.

The power of the Cshell, for instance, comes from the fact that filters (functions that read from the file descriptor 0 and write to the file descriptor 1) can be plugged together using pipes, and their input (output) can be read from (written to) any named file by means of the operator < (respectively, >). The following convention is adopted in the Gshell: for a Unix command (or, more generally, a C program) to be executable as a node in a tableau its input file descriptors must be consecutive even numbers (0, 2, 4, etc.) and its output must be directed to odd file descriptors (1, 3, 5, etc.). This has the advantage that existing Unix filters already obey this convention, but has the disadvantage that file descriptor 2 can no longer be used for standard error. The Gshell redirects standard error to file descriptor 19 (the largest file descriptor available on the system on which it is currently installed). This is not too much help, however, to processes that expect it to be 2.

The difficulty for the implementor of the scheme just described is that the Gshell system cannot provide processes with correctly numbered file descriptors, as it may, at any point, be in the middle of initiating a number of distinct processes all of which use the same numbering convention applied to different pipes and files. Added to this is the fact that the programmer has little control over the values of the file descriptors returned by open, creat, and pipe. For example, in the following diagram, after initiating process A, distinct pipes must be provided for processes B and C. At execution time, however, both of these processes will refer to the "read" end of their pipes by the same name, 0.



The solution to this problem is to let the Gshell pass to each command process the file descriptors which are returned by open, creat, and pipe. These file descriptors are actually passed in the filenum field of the corresponding tab of the node describing the command. When the process associated with a command is forked, the first thing it does is to reassign the file descriptors for all the files and pipes opened for it by the Gshell. This it does using the Unix function dup2 mentioned previously. After this is complete, the function execl is used to invoke the command associated with the node.

It is important to observe that the Gshell, itself, must close all file descriptors after it creates the child processes requiring them. If it were not to do this, two problems would arise. First, the Gshell could run out of new file descriptors. This is a serious problem in practice. The Gshell maintains 8 file descriptors permanently open while executing a tableau, in order to handle input and output to the screen. This leaves 12 file descriptors available, out of a total of 20, for the generation of pipes and the opening of files for other processes. The second problem arises if the file descriptor is the write end of a pipe. Any process reading this pipe will hang up for ever, waiting for input from a pipe it perceives as remaining open, while the Gshell itself waits in vain for the process to die.

Waiting for dead child processes is a housekeeping chore (if this term can be applied to such a macabre duty) that the Gshell must perform with complete thoroughness. If each child of a given tableau execution is not carefully laid to rest, it comes back to haunt the system the next time the EXEC key is pressed. To understand the reason for this, assume that after the execution of a tableau one child process remains dead but not laid to rest by the wait function. The next time a tableau is executed, the first wait executed by the Gshell will uncover this process and not one belonging to the current execution. Of course, the system can be designed to ignore unrecognizable processes, but the design will be cleaner if such processes are laid to rest in an organized fashion.

IV. FUTURE DEVELOPMENTS AND FINAL COMMENTS

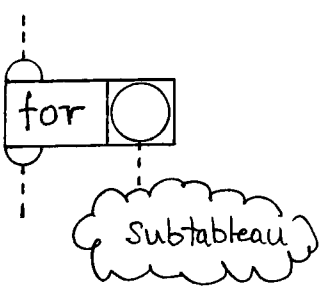
The version of the Gshell discussed in the preceding chapters represents only a partial realization of the original concept, limited principally by the desire to produce a working system within a reasonable amount of time. This section presents some of the features which would be seriously considered for incorporation into a future enhancement of this system.

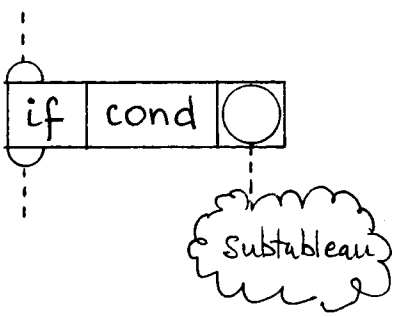
The full Gshell is a more powerful programming language embedded in a richer environment. The principal augmentation of the language itself concerns the addition of control structures, analogues of the conditional and looping structures of other programming languages. The enhancements to the environment include additional editing features and the incorporation of the directory structure of Unix into the Gshell page structure. These features are discussed in more detail below.

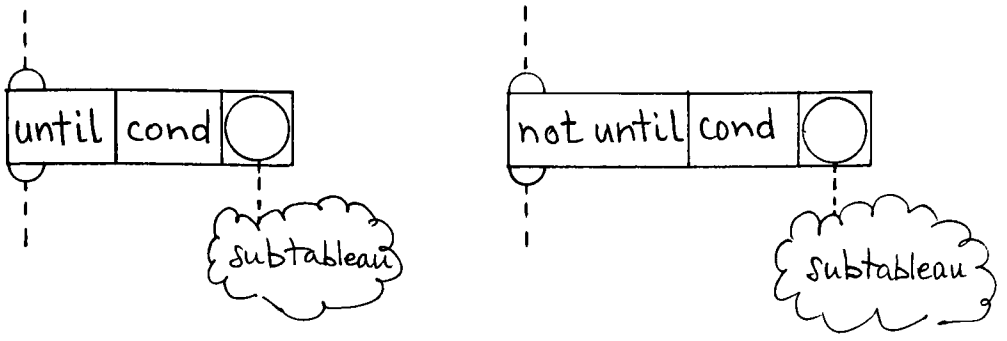
1. Control.

The Gshell, as currently implemented, does not contain any of the control structures which make the other shells so powerful. Its relatively simple structure could be modified by the introduction of control nodes, which permit the execution of subtableaux under conditions specified in the text of these nodes.

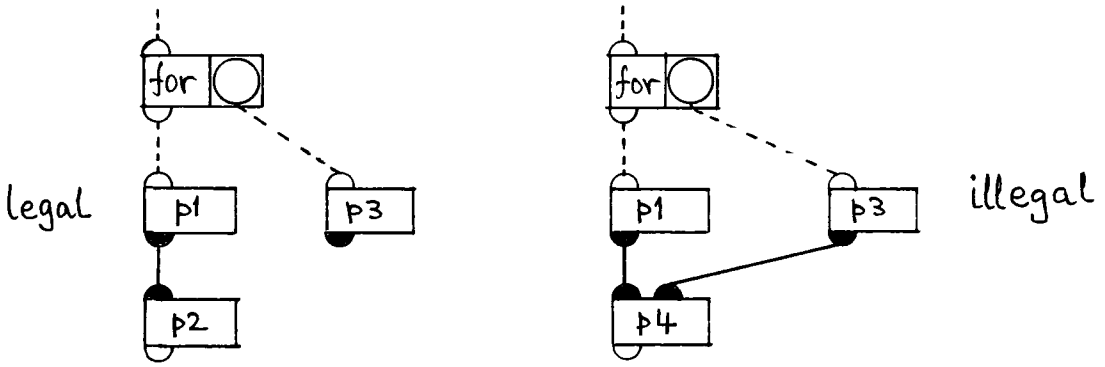
The following control nodes would be provided:





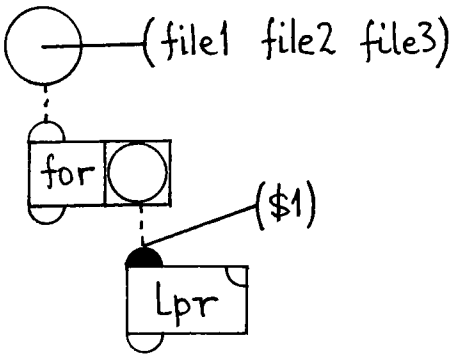


Each one controls the execution of a subtableau disjoint from the main tableau, except where it is connected via the control node, itself.



Execution of a control node is not complete until the execution of its associated subtableau is done.

The "for" node: This node converts the input list of the parent tableau into a sequence of inputs to its subtableau. This sequence causes repeated execution of the latter, once for each of the inputs in the sequence. For example, this



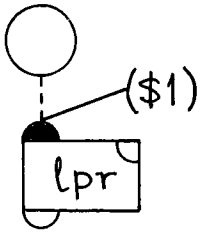
will cause first file1, then file2, and finally file3 to be listed on the line printer. That is, the input list

(file1 file2 file3)

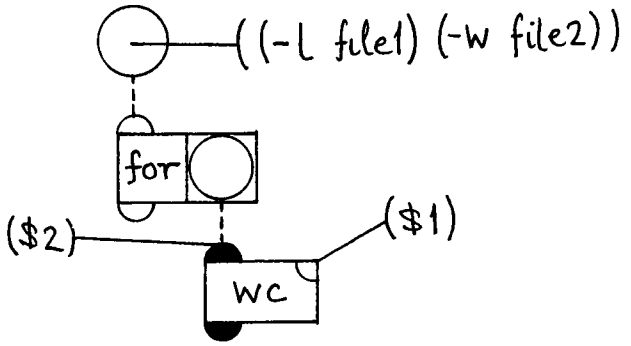
is converted to three separate input lists

(file1), (file2), (file3)

which cause three separate executions of the subtableau



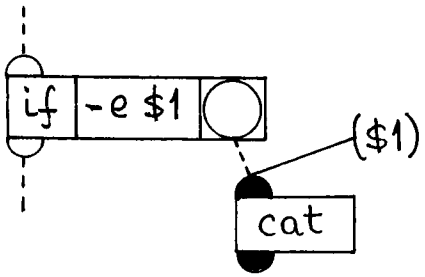
Sublists should be passed intact by "for" if they are enclosed in parentheses, e.g.



is equivalent to

wc -l file1; wc -w file2

The "if" node: This passes the parameters of the main tableau to its subtableau and initiates the execution of the latter only if the parameter list satisfies the condition specified, e.g.



is equivalent to

```
if (-e $1) cat $1
```

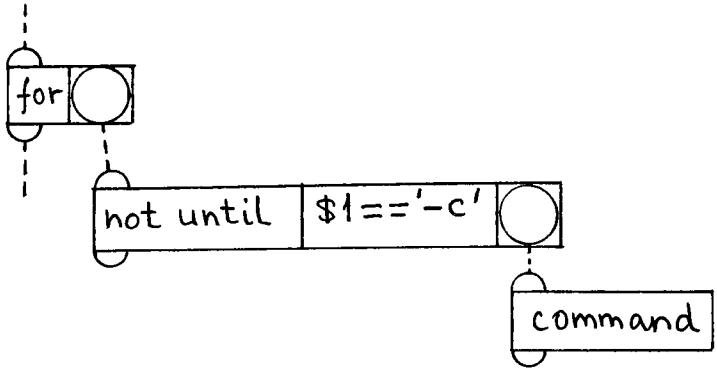
Note that the syntax of the conditional expression is similar to that allowed in the C-shell.

The "until" and "not until" nodes: These are partial substitutes for the variables of other shells, particularly when they are used as switches. For example,

```
for i

    if ($i=='-c') set enable
    if ($? enable) command
```

becomes



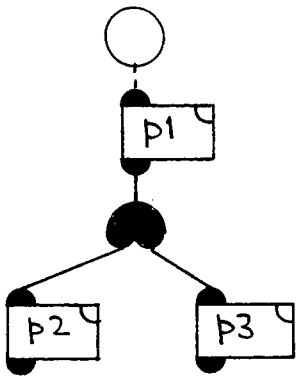
The "until" box passes parameter lists to its subtableau and executes each one until the specified condition is satisied. Thereafter, it takes no action.

On the other hand, "not until" does not activate the subtableau until the condition is satisfied. Once the subtableau has been switched on in this way, each parameter list is passed to it and executed before the process box following the "until" box commences execution of that same list

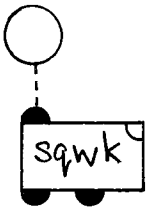
Note: For convenience, while and not while could also be provided.

2. The Environment.

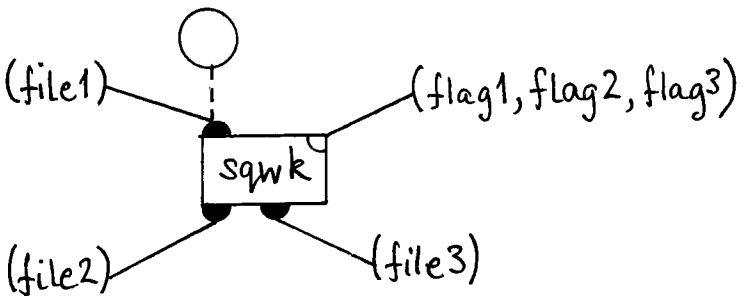
Facilities would be provided for the naming and saving of tableaux. A saved tableau could then be recalled for execution or to be incorporated into other tableaux.



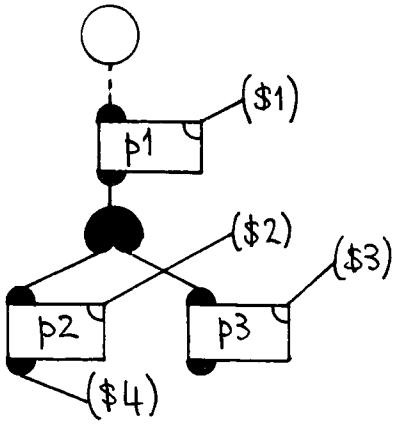
on being named, say, sqwk, this is represented thus



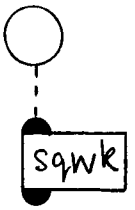
and is used in the form



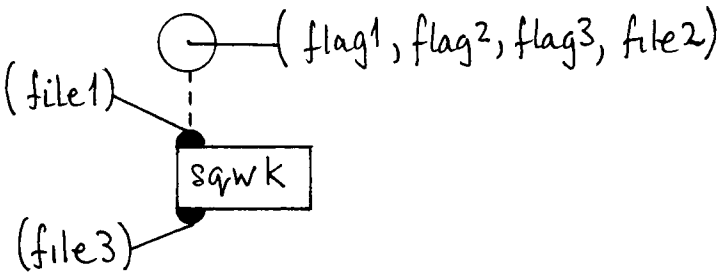
On the other hand, in this form:



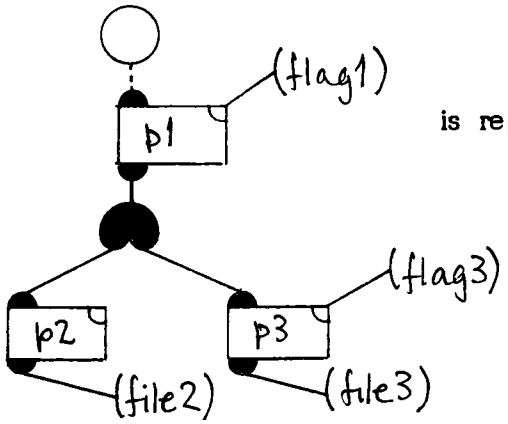
it is represented as



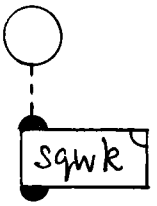
and used in the form



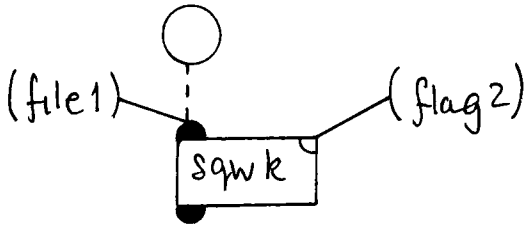
In a similar fashion



is represented as



and used in the form



Text editing facilities would also be provided to allow the user to modify the contents of data panels and pages containing text.

A modification of the EXEC command, corresponding to the "&" operator of other shells, would allow a process to run without requiring the user to wait for it. Currently, the control of the execution process (including the assignment of pipes and files, the forking of processes, etc.) is handled by the Gshell command interpreter, itself. This modification would require that control of execution be run as a child process of the Gshell, which could then continue to process other Gshell editing commands. There are, however, some subtleties which would provide some challenging programming problems. For example, if the user should change pages while an executing tableau is updating the current page, then output to the page would have to stop. This would require that the Gshell communicate with certain children of the execution controller. Such communication (of a process with its grandchildren) is not easily implemented under Unix. A third modification that, from the present perspective, appears to be a desirable feature would be the association of a Unix directory with each Gshell page, so that as the user switches from one page to another he or she also changes directories. This would allow the user to work within different directories on different pages without the conscious effort of changing directories when changing pages. There are two requirements if this is to be an aid and not an irritation. First, the user must be able to switch the feature on and off. Secondly, another field must be added to the page display showing the current directory for that page.

3. The Gshell as a Useable Tool.

It is clear that, for the Gshell to be a truly versatile and attractive programming tool, it must provide all the facilities of a more conventional command interpreter as well as additional concurrent programming facilities. Furthermore, it must be as convenient to use as any system it would replace.

Whether the Gshell is actually more convenient to use than, say, the Cshell is debatable. It certainly provides some features that the Cshell does not have but, on the other hand, like any limited experimental system, it lacks many features that most users would consider absolutely necessary. However, the goal of this project was not so much to produce a useable product as to experiment with a graphical approach to programming language syntax. A primary impetus for the project was the proliferation of languages providing built-in facilities for concurrent programming and a desire to see if a two dimensional graphical language might provide a clearer syntax for describing concurrency. The choice of "yet another Unix shell" as a vehicle for the project pays homage to the power and versatility of the Unix system, particularly in regard to its symbiotic relationship with the language C. Unix provided a convenient workbench on which to test the concept of a graphical description of concurrency.

The experience of working on a project of this nature does help to clarify some of the problems inherent in the design of a graphical user interface. First, the cliche, "a picture is worth a thousand words", is true for one very good reason. Pictures that <u>are</u> worth a thousand words contain a lot of information, in the technical sense, i.e., many pixels of information. All too many present day displays are really only toys. Their resolution is too low to be truly useful. A minimal requirement for a graphics terminal

is 1024 by 1024 pixels, in this author's opinion. Anything less severely limits the amount of information that can be displayed. The only alternative is to provide a two dimensional analogue of the scrolling required to handle text. The terminal screen should be a window into a much larger virtual field, which the user can shift left/right and up/down in order to view different parts of the display.

A second lesson, in some ways related to the first, is that the typical link between computer and terminal (e.g. an RS232 line) has too narrow a bandwidth. Certainly, at 9600 baud the updating of a screen is satisfyingly fast, but this still presumes a relatively simple display on a low resolution screen. Of course, here we meet yet another bottleneck: the average computer is still not yet powerful enough to maintain and update complex dislays in real time. It is here that parallelism at the hardware level offers interesting possibilities. One can easily imagine a system in which different sections of the screen are maintained by separate processors.

One final comment concerning the problem of graphical communication. The key-controlled cursor is a concept about as efficient as the punched card. This is one feature of the Gshell which limits its practical application. The selection of objects on the page is made slow and cumbersome by the simple necessity of getting the cursor from one place to another. The limitation of cursor movement to the horizontal and vertical directions makes this an onerous chore. The incorporation of support for a light pen or mouse is an absolute necessity if the Gshell is to be anything but a curiosity.

V. REFERENCES.

1. Ashcroft, E. A., and Wadge, W. W., "Lucid, a Nonprocedural Language with Iteration." Comm. ACM., Vol. 20, No. 7, Jul. 1977, pp 519-526.

2. Borning, A., "The Programming Language Aspects of ThingLab, a Constraint-Orientated Simulation Laboratory." ACM. Trans. Program. Lang. Syst., Vol. 3, No. 4, Oct. 1981, pp 353-387.

3. Bourne, S. R., "An Introduction to the UNIX shell." Bell Labs., 1978.

4. Davis, A. L., "Data Driven Nets: A Maximally Concurrent, Procedural, Parallel Process Representation for Distributed Control Systems.", Technical Report, Comp. Sci. Dept., Univ. of Utah, 1978.

5. Davis, A. L., and Keller, R. M., "Data Flow Program Graphs." Computer, Vol. 15, No. 2, Feb. 1982, pp 26-41.

6. Henderson, P., Functional Programming: Application and Implementation. Prentice-Hall, Englewood Cliffs, NJ, 1980.

7. Joy, W., "An Introduction to the C shell." UCLA, Berkeley, 1979.

8. Keller, R. M., and Yen, W-C. J., "A Graphical Approach to Software Development Using Function Graphs." Digest of Papers, Compcon Spring 81, Feb. 1981, pp 156-161.

9. Kernighan, B., and Richie, D. M., "UNIX Programming - Second Edition." Bell Labs., 1978.

10. McGraw, J. R., "The VAL Language: Description and Analysis." ACM. Trans. Program. Lang. Syst., Vol. 4, No. 1, Jan. 1982, pp 44-82.

11. Richie, D. M., "The UNIX Time-Sharing System - A Retrospective." Tenth Hawaii Internat. Conf. Syst. Sci., Honolulu, Jan. 1977.

12. Tesler, L., "The Smalltalk Environment." Byte, Vol. 6, No. 8, Aug. 1981, pp 90-147.

13. van den Bos, J., Plasmeijer, R., and Stroet, J., "Process Communication Based on Input Specifications." ACM. Trans. Program. Lang. Syst., Vol. 3, No. 3, Jul. 1981, pp 224-250.

14. Wolfberg, M. S., "Fundamentals of the AMBIT/L List Processing Language." ACM. SIGPLAN Notices, Vol. 7, No. 10, Oct. 1972, pp 66-75.

15 Byte: The Small Systems Journal, Vol. 6, No. 8, Aug. 1981.

VI. APPENDIX: SYNTAX DIAGRAMS FOR GSHELL COMMANDS.

The formal syntax of the Gshell command structure is as follows:

<Gshell_command> ::= <move_cursor> | <move_obj> | <window> |
<delete_obj> | <insert_obj> | <refresh> |
<page> | <edit_link> | <execute> | <exit>

<move_cursor> ::= ([↑] | [↓] | [←] | [→])*

<move_obj> ::= [MOV] <move_cursor> [END]

<window> ::= [WIND] <move_cursor> [END]

<delete_obj> ::= [DEL]

<refresh> ::= [REF]

<page> ::= [PAGE]

<insert_obj> ::= [INS] (<unnamed_obj> | <named_obj> | [END])

<edit_link> ::= <select_link>(<insert_link> | <delete_link> |
<select_link>)* [END]

<execute> ::= [EXEC]

<exit> ::= CTRL-C

<unnamed_obj> ::= [TEE] | [DPAN]

<named_obj> ::= ([CMD] | [PARAM])<name> [END]

<name> ::= (<character>)*

<character> ::= Any printable ASCII character

<select_link> ::= [SEL] ([SEL])*

<delete_link> ::= [DEL]

<insert_link> ::= [INS] <move_cursor><select_link>
([CHAN] | [PIPE] | [END])

VII. APPENDIX: SOURCE CODE FOR GSHELL SYSTEM.

The main procedure is EXECUTE GSHELL. The first pages of this listing contain system-wide constants and definitions. Thereafter, the procedures are listed alphabetically.

```
        /*- - - - - - - - - - - - - - - - - - - - - - - - - */
        /*  GSHELL CONSTANT, MACRO                         */
        /*  AND TYPE DEFINITIONS                           */
        /*- - - - - - - - - - - - - - - - - - - - - - - - - */

#include <stdio.h>
#include <ctype.h>

        /* Memory Requirements */

#define MAXFDS          20      /* max number of files allowed by UNIX */
#define ONODEMAX        64
#define ONODESIZE       sizeof(struct onodetype)
#define PNODESIZE       sizeof(struct pnodetype)
#define PRCNODESIZE     sizeof(struct procnodetype)
#define ARGNODESIZE     sizeof(struct arglisttype)
#define TABMAX          6
#define FIRSTTAB        1
#define NAMEMAX         80
#define NKEYS           8
#define NKEYSMN1        NKEYS - 1
#define NPAGES          8
#define NARGS           16

        /* Key Codes and Object Classes (ANSI terminal mode) */

#define COMMAND         'u' + 128       /* ANSI Code: ESC O u */
#define PIPE            'q' + 128       /* etc. */
#define CHANNEL         'r' + 128
#define IIF             's' + 128       /* not used */
#define UNTIL           'Y' + 128       /* not used */
#define NOTUNTIL        'Z' + 128       /* not used */
#define FFOR            'n' + 128       /* not used */
#define TEE             't' + 128
#define START           'm' + 128
#define DPANEL          'v' + 128
#define PARAM           'l' + 128
#define MOVE            'Q' + 128
#define INSERT          'P' + 128
#define WINDOW          'R' + 128
#define DELETE          'S' + 128
#define REFRESH         'w' + 128
#define PAGE            'y' + 128
#define SELECT          'x' + 128
#define EXECUTE         'M' + 128
#define END             'p' + 128
#define UPCURSOR        'A' + 128       /* ANSI Code: ESC [ A */
#define DOWNCURSOR      'B' + 128       /* etc. */
#define RIGHTCURSOR     'C' + 128
#define LEFTCURSOR      'D' + 128
#define SPACECHAR       ' '
#define TABCHAR         '        '
#define NEWLINE         '\n'
```

```
#define ATCHAR          '@'
#define TILDECHAR       '~'
#define MAXASCII        '\177'
#define MAXASCPL1       128
#define EXITCHAR        '\003'          /* CTRL C, for now at least */
#define ESCCHAR         '\033'
#define CTRLT           '\024'
#define CTRLN           '\016'
#define BACKSPACE       '\b'
#define NULLCHAR        '\000'

        /* Tab Classes */

#define INTAB           'i'
#define INSYNCTAB       's'
#define OUTTAB          'o'
#define OUTSYNCTAB      't'
#define FLAGTAB         'f'
#define DATATAB         'd'
#define NOTUSED         'n'

        /* Status Flags */

#define UNBORN          -1
#define ALIVE           0
#define DEAD            1
#define TABLEAU         0
#define TEXT            1

        /* File Constants */

#define NOFILE          -1
#define READ            0
#define WRITE           1
#define NORMALMODE      0644

        /* Graphics Flags */

#define DRAW            'V'
#define ERASE           'E'
#define SOLID           '1'
#define DOTTED          '2'
#define CURSLINSTYLE    '4'
#define BLINKOFF        '0'
#define BLINKON         '1'
#define SHADEOFF        '0'
#define SHADEON         '1'
#define COMPLMNT        'C'
#define REVERSE         '1'
#define NORMAL          '0'

        /* Cursor Codes */
```

```
#define ARROW           1
#define ANGLE           2
#define CROSS           3
#define BLOCK           4
#define GIGICURSOR      5

        /* Dimensions & Coordinates of Graphic Objects */

#define XMESH           48
#define YMESH           48
#define XGRIDMAX        16
#define YGRIDMAX        10
#define XPAGELIM        15
#define YPAGELIM        9
#define XPAGE0          720
#define YPAGE0          48
#define YPAGE7          384
#define XMIDSCR         352
#define YMIDSCR         208
#define XINF            16
#define YINF            32
#define XSUP            704
#define YSUP            432
#define BORDERWD        16
#define BORDERWD2       2 * BORDERWD
#define PAGEHT          400
#define PAGEWD          688
#define OUTFRMHT        432
#define OUTFRMPL16      OUTFRMHT + 16
#define HMARGIN         8
#define VMARGIN         6
#define HMARGIN2        2 * HMARGIN
#define CHARHT          10
#define CHARHT2         2 * CHARHT
#define CHARHT4         4 * CHARHT
#define CHARWD          8
#define CHARWD2         2 * CHARWD
#define CHARWD3         3 * CHARWD
#define CHARWD4         4 * CHARWD
#define TABRAD          8
#define TABRAD2         2 * TABRAD
#define LINESEPN        20
#define TABSEPN         32
#define CHARSEPN        9
#define STARTRAD        16
#define PTABRAD         16
#define XMESSBEGIN      112
#define YMESSBEGIN      456
#define MESSLEN         592
#define XCMDBEGIN       16
#define YCMDBEGIN       456
#define CMDLEN          80
#define XGIGIHI         767
```

```
#define YGIGIHI          479
#define XPNUMOFFST       16
#define YPNUMOFFST       6
#define PNODESEPN        48
#define PNODEWD          47
#define BOXHT            32
#define  WSTDPANEL       128
#define HSTDPANEL        64
#define DELTA            16
#define ORDPAGE0         990
#define ORDPAGE7         997

        /* Other Constants */

#define X                0
#define Y                1
#define OK               0
#define NOTOK            1
#define FALSE            0
#define TRUE             1
#define KILLED           9

        /* Macro Definitions */

#define GIGICURSORON  "\033PrVC2\033\\\033Pp"
#define GIGICURSOROFF "\033PrVC0\033\\\033Pp"
#define printposn(A,B)      printf("P[%d,%d]",A,B)
#define iscursor(A)         ((A)==UPCURSOR || (A)==DOWNCURSOR || \
                            (A)==LEFTCURSOR || (A)==RIGHTCURSOR)
#define iskeypad(A)         ((A)==COMMAND || (A)==IIF || (A)==UNTIL || \
                            (A)==FFOR || (A)==NOTUNTIL || (A)==TEE || \
                            (A)==START || (A)==MOVE || (A)==DPANEL || \
                            (A)==PARAM || (A)==INSERT || (A)==END || \
                            (A)==WINDOW || (A)==DELETE || (A)==REFRESH || \
                            (A)==PAGE || (A)==SELECT || (A)==EXECUTE || \
                            (A)==PIPE || (A)==CHANNEL)
#define min(A,B)            ((A)<(B) ? (A) : (B))
#define max(A,B)            ((A)>(B) ? (A) : (B))
#define isonpage(A,B)       (XINF<(A) && (A)<XSUP && YINF<(B) && (B)<YSUP)
#define isnotobject(A)      ((A)!=COMMAND && (A)!= IIF && (A)!=UNTIL &&\
                            (A)!=NOTUNTIL && (A)!=PARAM && (A)!=FFOR &&\
                            (A)!=START && (A)!=TEE && (A)!=DPANEL)
#define isnameable(A)       ((A)==COMMAND || (A)==IIF || (A)==UNTIL ||\
                            (A)==NOTUNTIL || (A)==PARAM)
#define isdataobj(A)        ((A)==PAGE || (A)==DPANEL || (A)==PARAM)
#define isintab(A)          ((A)==INTAB || (A)==INSYNCTAB)
#define isouttab(A)         ((A)==OUTTAB || (A)==OUTSYNCTAB)
#define areintabs(A,B)      (isintab(A) && isintab(B))
#define areouttabs(A,B)     (isouttab(A) && isouttab(B))
#define issynctab(A)        ((A)==INSYNCTAB || (A)==OUTSYNCTAB)
#define iswhitespace(A)     ((A)==SPACECHAR || (A)==TABCHAR || (A)==NEWLINE)
#define hasinchan(O,I)      (isintab(O->tab[I].tclass) && (O->tab[I].clink!=NULL)\
                            && (O->tab[I].clink!=O->tab[I].dlink))
```

```
#define hasoutchan(O,I)    (isouttab(O->tab[I].tclass) && (O->tab[I].clink!=NULL)\
                           && (O->tab[I].clink!=O->tab[I].dlink))
#define haspipe(O,I)       ((O->tab[I].clink!=NULL) \
                           && (O->tab[I].clink==O->tab[I].dlink))

#define visible(A,B,R)     (XINF<(A) && YINF<(B) && (A+R)<XSUP && (B+R)<YSUP)

        /* Type Definitions */

struct onodetype {                          /* OBJECT NODE */
  int ord;                                  /* ordinal value */
  int oclass;                               /* object class */
  int status;                               /* node status during execute */
                                            /* and page status */
  int drawflag;                             /* used to draw page */
  char name[NAMEMAX];                             /* object name */
  int x,y;                                  /* object location */
  int wr,h;                                 /* width or radius, height */
  struct {                                  /* tab descriptors */
    char tclass;                            /* tab class */
    int filenum;                            /* file number during execute */
    struct onodetype *clink;                /* ptr for control link */
    int ctab;                               /* tab number of dest. */
    struct onodetype *dlink;                /* ptr for data link */
    int dtab;                               /* tab number of dest. */
  } tab[TABMAX];
};

struct pnodetype {                          /* POINTER NODE */
  struct onodetype *obj;                    /* pointer to object node */
  struct pnodetype *next;                   /* pointer to next ptr node */
};

struct procnodetype {                       /* PROCESS NODE */
  int process;                              /* process number */
  struct onodetype *object;                 /* pointer to object node */
  struct procnodetype *nextproc;            /* pointer to next process node */
};

typedef struct procnodetype *proclisttype;  /* pointer to process list ptr */

struct arglisttype {                        /* ARGUMENT LIST IN EXECUTE */
  char argstring[NAMEMAX];                  /* string of chars. in argument */
  int begin[NARGS];                         /* pointers to individ. args. */
  int number;                               /* number of arguments */
};
```

```
            k = (int)optr->tab[j].clink;
            optr->tab[j].clink = onodelist[k];
            k = (int)optr->tab[j].dlink;
            if (0<=k && k<ORDPAGE0){
                optr->tab[j].dlink = onodelist[k];
            }
            else if (k<=ORDPAGE7){
                loc[X] = XPAGE0; loc[Y] = YPAGE0+(k-ORDPAGE0)*PNODESEPN;
                optr->tab[j].dlink = pageptr = getobj(loc,&notused);
                pageptr->tab[0].dlink = optr;
                pageptr->tab[0].dtab = j;
            }
            else {
                printmess("BAD LINK FROM PAGE FILE");
                return(1);
            }
        } /* for j */
    } /* if */
} /* for i */
return(0);
}
```

```
        /*- - - -  - - - - - - - - - -  -  */
        /*  C O P Y   P A G E   T O   F I L E  */
        /*- - - - - -  - - - - - - - - - - - */
/*
          Action: If current page is a tableau page, then
                  remove objects from page map and write
                  them in the page file. Objects are
                  numbered (ord) and tab links are replaced
                  by the (ordinal) numbers of their
                  destination.
                  If page is a text page, do nothing.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern struct pnodetype *pagemap[YGRIDMAX][XGRIDMAX];
extern struct onodetype *currpage;

copyptof(pagename)

char pagename[];

{
   FILE *fp,*fopen();
   int i,j,k;
   char class,status;
   struct onodetype *optr,*dptr,*onodelist[ONODEMAX];
   struct pnodetype *pptr,*oldptr;

   /* DEBUG */
   if (debug) printf("** copyptof **");

   if (currpage->status==TEXT)
      return(0);

   /* assemble list of all object nodes to be copied to page file */
   k = 0;
   for (i=0; i<XPAGELIM; i++){
      for (j=0; j<YPAGELIM; j++){
         pptr = pagemap[j][i]; pagemap[j][i] = NULL;
         while (pptr != NULL){
            if (++k == ONODEMAX){
               printmess("TOO MANY NODES: KILL SOME");
               return(1);
            } /* if */
            onodelist[k] = pptr->obj;
            pptr->obj->ord = k;
            oldptr = pptr;
            pptr = pptr->next;
            removefromap(oldptr->obj);
```

```
        } /* while */
    } /* for j */
} /* for i */

/* if page not empty, copy list of object nodes to the page file */
fp = fopen(pagename,"w");
if (k>0){
    status = CTRLT; fprintf(fp,"%c ",status);
    for(i=1; i<=k; i++){
        optr = onodelist[i];
        class = optr->oclass & 127;
        fprintf(fp,"%d %c %s%c %d %d %d %d ",
                i,
                class,
                optr->name,CTRLN,
                optr->x, optr->y,
                optr->wr, optr->h);
        for (j=0; j<TABMAX; j++){
            dptr = optr->tab[j].dlink;
            if (dptr->oclass==PAGE){
                dptr->tab[0].dlink = NULL;
                dptr->tab[0].dtab = 0;
            }
            fprintf(fp,"%c %d %d %d %d ",
                    optr->tab[j].tclass,
                    optr->tab[j].clink==NULL ? 0 : optr->tab[j].clink->ord,
                    optr->tab[j].ctab,
                    optr->tab[j].dlink==NULL ? 0 : optr->tab[j].dlink->ord,
                    optr->tab[j].dtab);
        } /* for j */
    } /* for i */

    /* dispose of list of object nodes */
    for (i=1; i<=k; i++){
        if (onodelist[i] != NULL){
            cfree((char *)onodelist[i]);
            onodelist[i] = NULL;
        }
    }
}
fclose(fp);
return(0);
}
```

```
        /*- - - - - - - - */
        /*  C U R S O R   */
        /*- - - - - - - - */
/*
            Action: Cursor of given style is drawn or erased
                    at the given location.
            Note:   The GIGI terminal graphics cursor is a
                    special case. It is drawn merely by
                    specifying a location.

*/

#include "Gshell.defs"

extern int debug;

cursor(loc,cursorstyle,mode)

int loc[2],cursorstyle;
char mode;

{
    char *buff,*calloc();

    /* DEBUG */
    if (debug) printf("** cursor **");

    buff = calloc(16,sizeof(char));

    /* GIGICURSOR is handled separately. This is the built in graphics */
    /* cursor. It is used in the Gshell as the general purpose cursor */
    /* because it is fast. */
    if (cursorstyle==GIGICURSOR){
        if (mode==DRAW){
            sprintf(buff,"P[%d,%d]",loc[X],loc[Y]);
            write(1,buff,strlen(buff));
            cfree(buff);
            return(0);
        }
        else
            cfree(buff);
            return(0);
    }

    /* all other cursor styles are handled here */
    if (mode == DRAW){
        sprintf(buff,"@N");
        write(1,buff,strlen(buff));
    }
    else if (mode == ERASE){
        sprintf(buff,"@O");
        write(1,buff,strlen(buff));
    }
```

```
    else
        printmess("INVALID CURSOR MODE");

    /* position at current location */
    sprintf(buff,"P[%d,%d]",loc[X],loc[Y]);
    write(1,buff,strlen(buff));

    /* draw cursor */
    switch (cursorstyle){
    case ARROW:
        sprintf(buff,"P[-%d]@A",CHARWD2);
        write(1,buff,strlen(buff));
        break;
    case CROSS:
        sprintf(buff,"P[-%d,-%d]@B",CHARWD,CHARHT);
        write(1,buff,strlen(buff));
        break;
    case ANGLE:
        sprintf(buff,"P[-%d,-%d]@C",CHARWD2,CHARHT2);
        write(1,buff,strlen(buff));
        break;
    case BLOCK:
        sprintf(buff,"@D");
        write(1,buff,strlen(buff));
        break;
    default:
        printmess("INVALID CURSOR TYPE");
        break;
    }
    sprintf(buff,"@P");
    write(1,buff,strlen(buff));
    cfree(buff);
    return(0);
}
```

```
                /*- - - - - - - */
                /*  A B O R T  */
                /*- - - - - -  */
/*
                    Action: Kill all outstanding command node processes.
                            Close all file descriptors. Wait for all
                            children to die.

*/

#include "Gshell.defs"

                /* globals */
extern int debug;

abort(ptrproclist)

proclisttype *ptrproclist;

{
    int i,status;
    struct procnodetype *procptr;

    /* DEBUG */
    if (debug) printf("** abort **");

    /* kill all command processes currently running */
    while (*ptrproclist!=NULL){
        procptr = *ptrproclist;
        kill(procptr->process);
        *ptrproclist = procptr->nextproc;
        cfree((char *)procptr);
    }

    /* close all files, except stdin, stdout, and stderr */
    for (i=3; i<20; i++)
        close(i);

    /* wait for all children to die */
    while (wait(&status)>=0) ;

    return(0);
}
```

```
        /*- - - - - - - - - - - - - */
        /*  A D D   T O   M A P    */
        /*- - - -   - - - - - -    */
/*
            Action: Add pointers to object node from every grid
                    square that intersects its image on the
                    screen.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern int pagedrawflag;
extern struct pnodetype *pagemap[YGRIDMAX][XGRIDMAX];

addtomap(optr)

struct onodetype *optr;

{
   int i,j,low[2],high[2];
   char *calloc();
   struct pnodetype *pptr;

   /* DEBUG */
   if (debug) printf("** addtomap **");

   optr->drawflag = pagedrawflag;
   getextentobj(optr,low,high);

   /* add pointer to object at each grid square of */
   /* pagemap which intersects object */
   for (i=low[X]; i<=min(high[X],XPAGELIM); i++){
      for (j=low[Y]; j<=high[Y]; j++){
         if ((pptr = (struct pnodetype *)calloc(1,PNODESIZE)) == NULL){
            printmess("NO MEM. SPACE FOR NODE");
            return(1);
         }
         pptr->obj = optr;
         pptr->next = pagemap[j][i];
         pagemap[j][i] = pptr;
      } /* for j */
   } /* for i */
   return(0);
}
```

```
        /*- - - - - */
        /*  B O X   */
        /*- - - - - */
/*
            Action: Draw or erase rectangular box on diagonal
                    specified by the high and low arrays

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

box(low,high,mode)

int low[2],high[2];
char mode;

{
    /* DEBUG */
    if (debug) printf("** box **");

    /* top */
    horizbar(low[X],high[X],low[Y],mode);
    /* bottom */
    horizbar(low[X],high[X],high[Y],mode);
    /* left */
    vertbar(low[X],low[Y],high[Y],mode);
    /* right */
    vertbar(high[X],low[Y],high[Y],mode);
}


        /*- - - - - - - - -     - - - -   */
        /*  H O R I Z O N T A L   B A R   */
        /*- - -   - - - -   - - - - - - - */
/*
            Action: Draw horizontal line segment between
                    given points.

*/

horizbar(x1,x2,y,mode)

int x1,x2,y;
char mode;

{
    char *buff,*calloc();

    /* DEBUG */
    if (debug) printf("** horizbar **");
```

```
    if (y>YINF && y<YSUP && x1<XSUP && x2>XINF){
        buff = calloc(32,sizeof(char));
        sprintf(buff,"P[%d,%d]@J%c))[%d,%d]",max(x1,XINF),y,mode,min(x2,XSUP),y);
        write(1,buff,strlen(buff));
        cfree(buff);
    }
}


        /*-   - - - - - - - - - - - - - */
        /*  V E R T I C A L   B A R   */
        /*- - - -   - - - -   - - - - */
/*
            Action: Draw vertical line segment between given
                    points.

*/

vertbar(x,y1,y2,mode)

int x,y1,y2;
char mode;

{
    char *buff,*calloc();

    /* DEBUG */
    if (debug) printf("** vertbar **");

    if (x>XINF && x<XSUP && y1<YSUP && y2>YINF){
        buff = calloc(32,sizeof(char));
        sprintf(buff,"P[%d,%d]@J%c))[%d,%d]",x,max(y1,YINF),mode,x,min(y2,YSUP));
        write(1,buff,strlen(buff));
        cfree(buff);
    }
}
```

```
        /*- - - - - - - - - - - - - - - */
        /*  B U R Y   P R O C E S S    */
        /*- - - -   - - -   - - - - - - */
/*
            Action: Find given process in process list, mark
                    it as dead and remove it.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

struct onodetype *buryprocess(procid,ptrproclist)

int procid;
proclisttype *ptrproclist;

{
   struct procnodetype *oldprocptr,*procptr;
   struct onodetype *optr;

   /* DEBUG */
   if (debug) printf("** buryprocess **");

   /* find process list entry for this process */
   oldprocptr = NULL;
   procptr = *ptrproclist;
   while (procptr!=NULL && procptr->process!=procid){
      oldprocptr = procptr;
      procptr = procptr->nextproc;
   }

   /* couldn't find it */
   if (procptr==NULL)
      return(NULL);

   /* found it, mark its object node as DEAD */
   procptr->object->status = DEAD;
   /* remove entry from process list */
   if (oldprocptr==NULL)
      *ptrproclist = procptr->nextproc;
   else
      oldprocptr->nextproc = procptr->nextproc;
   /* and dispose of it */
   optr = procptr->object;
   cfree((char *)procptr);
   return(optr);
```

```
        /*- - - - - - - - - - - */
        /*  CLEAR  MAP  */
        /*- - - - - - - - - - - */
/*
            Action: Remove all pointers to objects from page
                    map.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern struct pnodetype *pagemap[YGRIDMAX][XGRIDMAX];

clearmap()

{
    int i,j;
    struct pnodetype *pptr,*oldptr;

    /* DEBUG */
    if (debug) printf("** clearmap **");

    for (i=0; i<XPAGELIM; i++)
        for (j=0; j<YPAGELIM; j++){
            oldptr = NULL;
            pptr = pagemap[j][i];
            pagemap[j][i] = NULL;
            while (pptr != NULL){
                oldptr = pptr;
                pptr = pptr->next;
                removefromap(oldptr->obj);
                cfree((char *)oldptr->obj);
            } /* while */
        } /* for j */
    return(0);
}
```

```
        /*- - - - - - */
        /*  C L I P   */
        /*- - - - - - */
/*
            Action: Clip specified line to boundaries of work
                    area of page.

*/

#include "Gshell.defs"
#define LEFT            1
#define RIGHT           2
#define TOP             4
#define BOTTOM          8
#define EMPTY           0
#define clipset(A,B)    (((A)<XINF)*LEFT+((A)>XSUP)*RIGHT+((B)<YINF)*TOP
                        +((B)>YSUP)*BOTTOM)

        /* globals */
extern int debug;

clip(p1,p2)

int p1[2],p2[2];

{
    int c,c1,c2,xdiff,ydiff,x,y;

    /* DEBUG */
    if (debug) printf("** clip **");

    c1 = clipset(p1[X],p1[Y]);
    c2 = clipset(p2[X],p2[Y]);
    while ((c1|c2)!=EMPTY && (c1&c2)==EMPTY){
        if ((c=c1)==EMPTY) c = c2;
        ydiff = p2[Y]-p1[Y]; xdiff = p2[X]-p1[X];
        if ((LEFT&c) != EMPTY){
            x = XINF; y = p1[Y] + ydiff*(x-p1[X])/xdiff;
        }
        else if ((RIGHT&c) != EMPTY){
            x = XSUP; y = p1[Y] + ydiff*(x-p1[X])/xdiff;
        }
        else if ((TOP&c) != EMPTY){
            y = YINF; x = p1[X] + xdiff*(y-p1[Y])/ydiff;
        }
        else if ((BOTTOM&c) != EMPTY){
            y = YSUP; x = p1[X] + xdiff*(y-p1[Y])/ydiff;
        }
        if (c == c1){
            c1 = clipset(x,y);
            p1[X] = x; p1[Y] = y;
        }
        else {
```

```
            c2 == clipset(x,y);
            p2[X] = x; p2[Y] = y;
        }
    }
    return((c1&c2) == EMPTY);
}
```

```
        /*- - - - - - - - - - - - - - - - - - - - - - - - */
        /*  C O P Y   P A G E   F R O M   F I L E   */
        /*- - - - - - - -       - - - -    - - - -      */
/*
            Action: Read in objects or text from page file
                    for current page. Create object node for
                    each object and place pointers to it in
                    page map. Set up links between objects
                    according to link descriptions in file.
                    If a text page, then clip text to work-
                    space boundaries.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern int pagedrawflag;
extern struct onodetype *currpage;

copypfromf(pagename)

char pagename[ ];

{
   FILE *fp,*fopen();
   int i,ordinal,locx,locy;
   char ch,*mess,*calloc();
   struct onodetype *optr,*onodelist[ONODEMAX];

   /* DEBUG */
   if (debug) printf("** copypfromf **");

   if ((fp=fopen(pagename,"r")) == NULL){
      mess = "CANNOT OPEN ";
      printmess2(mess,pagename);
      return(1);
   }

   /* determine whether page holds tableaux or text */
   if (fscanf(fp,"%c",&ch)!=EOF){
      currpage->status = (ch==CTRLT) ? TABLEAU : TEXT;
      if (currpage->status==TABLEAU){
         /* read objects from page file into onodelist and page map */
         for (i=0; i<ONODEMAX; i++) onodelist[i] = NULL;
         pagedrawflag = 0;
         while (fscanf(fp,"%d",&ordinal) != EOF){
            if ((optr=(struct onodetype *)calloc(1,ONODESIZE)) == NULL){
               printmess("NO MEM. SPACE FOR NODE");
               return(1);
            }
            onodelist[ordinal] = optr;
```

```
            fscanf(fp,"%ls",&optr->oclass);
            while ((ch=getc(fp))==' ');
            i = 0;
            do { optr->name[i++] = ch; } while ((ch=getc(fp))!=CTRLN);
            optr->name[i] = ' ';
            fscanf(fp," %d %d %d %d",
                    &optr->x, &optr->y,
                    &optr->wr, &optr->h);
            optr->oclass |= 128;
            optr->ord = 0; optr->drawflag = pagedrawflag;
            for (i=0; i<TABMAX; i++)
               fscanf(fp,"%ls %d %d %d %d",
                       &optr->tab[i].tclass,
                       &optr->tab[i].clink,
                       &optr->tab[i].ctab,
                       &optr->tab[i].dlink,
                       &optr->tab[i].dtab);
            if (optr->oclass==DPANEL)
               renamedpanel(optr);
            addtomap(optr);
         } /* while */
         putlinkptrs(onodelist);
      }
      else {
         locx = XINF; locy = YINF;
         printf("P[%d,%d]",locx,locy); fflush(stdout);
         printf("@IV))'%c'",ch); fflush(stdout);
         locx += CHARSEPN;
         while ((fscanf(fp,"%c",&ch)!=EOF) && (locy<YSUP)){
            if (ch==NEWLINE){
               locx = XINF; locy += LINESEPN;
               printf("P[%d,%d]",locx,locy); fflush(stdout);
            }
            else {
               if (locx>=XSUP){
                  locx = XINF; locy += LINESEPN;
                  printf("P[%d,%d]",locx,locy); fflush(stdout);
               }
               printf("@IV))'%c'",ch); fflush(stdout);
               locx += CHARSEPN;
            } /* else */
         } /* while */
      } /* else */
   } /* if */
   else
      currpage->status = TABLEAU;
   fclose(fp);
   return(0);
}
```

```
        /*- - - - - - - - - - - - - - - - - - - - */
        /*  R E N A M E   D A T A   P A N E L    */
        /*- - - -   -   - - - - -   - - - - - - - */
/*
            Action: Ensure that data panels copied from page
                    file refer to current page in their names.
                    This is necessary if contents of page
                    have been copied from another page.

*/

renamedpanel(optr)

struct onodetype *optr;

{
   int i;

   for (i=0; i<NAMEMAX-4; i++)
      if (optr->name[i]=='p' &&
        optr->name[i+1]=='a' &&
        optr->name[i+2]=='g' &&
        optr->name[i+3]=='e' &&
        optr->name[i+4]=='.' )  break;
   optr->name[i+5] = currpage->ord-ORDPAGE0+'0';
}


        /*- - - - - - - - -   -   - - - - - - - */
        /*  P U T   L I N K   P O I N T E R S    */
        /*- - - - - - - - - - - - - -   -     - */
/*
            Action: Translate ordinal numbers in tab link
                    fields to pointers.
            Note:   In page file, pointers are meaningless so
                    they are replaced by ordinal numbers.
                    That is, each object has a small integer
                    assigned to it.

*/

putlinkptrs(onodelist)

struct onodetype *onodelist[ONODEMAX];

{
   struct onodetype *optr,*pageptr,*getobj();
   int i,j,k,notused,loc[2];

   for (i=0; i<ONODEMAX; i++){
      if (onodelist[i] != NULL){
         optr = onodelist[i];
         for (j=0; j<TABMAX; j++){
```

```
        /*- - - - - - - - */
        /*  D E L E T E   */
        /*- - - - - -   - */
/*
            Action: If object is the current page, then erase
                    workspace area of screen, remove all
                    objects from page map, and delete page
                    file for this page.  Otherwise, erase
                    given object, remove it from the page map,
                    and remove all links pointing to it.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern struct onodetype *currpage;

delete(loc,optr)

int loc[2];
struct onodetype *optr;

{
   int low[2],high[2];
   char removefile[CMDLEN];

   /* DEBUG */
   if (debug) printf("** delete **");

   if (optr->oclass == PAGE){
      if (optr == currpage){
         /* erase just the workspace area of the page */
         printf("P[%d,%d]@RW(E)",XINF+2,YINF+2);
         printf("P[,%d]V[%d]@QW(V)",YSUP-2,XSUP-2);
         printf("P[%d,%d]",loc[X],loc[Y]);
         fflush(stdout);
         clearmap();
         if (fexists(currpage->name)){
            sprintf(removefile,"rm %s*",currpage->name);
            system(removefile);
         }
         currpage->status = TABLEAU;
      }
      else {
         printmess("CHOOSE CURRENT PAGE");
      }
   }
   else {
      drawobj(optr,ERASE);
      removelinks(optr);
      removefromap(optr);
```

```
        if (optr->oclass==DPANEL && fexists(optr->name)){
            sprintf(removefile,"rm %s",optr->name);
            system(removefile);
        }
        cfree((char *)optr);
    }
    return(0);
}
```

```
        /*- - - - - - -   -   - - - */
        /*  D E L E T E   L I N K   */
        /*-   - - - -     -   - - - */
/*
          Action: Remove specified link pointer from
                  specified tab. Remove the reverse pointer
                  from the destination tab. If control link
                  == data link from this tab, then delete
                  them both if either is chosen.

*/

#include "Gshell.defs"
#define CONTROL         0
#define DATA    1

        /* globals */
extern int debug;

deletelink(tabnum,plinknum,optr)

int tabnum,*plinknum;
struct onodetype *optr;

{
    int dellink,othertab,tab1[2],tab2[2];
    struct onodetype *cptr,*dptr,*otherptr;

    /* DEBUG */
    if (debug) printf("** deletelink **");

    cptr = optr->tab[tabnum].clink;
    dptr = optr->tab[tabnum].dlink;
    if (cptr==NULL && dptr==NULL){
        printmess("NOTHING TO DELETE");
        return(1);
    }
    dellink = CONTROL;
    if (dptr!=NULL && (cptr==NULL || *plinknum==1))
        dellink = DATA;
    if (dellink==CONTROL || cptr==dptr){
        otherptr = cptr;
        othertab = optr->tab[tabnum].ctab;
        optr->tab[tabnum].clink = NULL;
        optr->tab[tabnum].ctab = 0;
    }
    if (dellink==DATA || cptr==dptr){
        otherptr = dptr;
        othertab = optr->tab[tabnum].dtab;
        optr->tab[tabnum].dlink = NULL;
        optr->tab[tabnum].dtab = 0;
    }
    if (otherptr->tab[othertab].clink == optr){
```

```
        otherptr->tab[othertab].clink = NULL;
        otherptr->tab[othertab].ctab = 0;
    }
    if (otherptr->tab[othertab].dlink == optr){
        otherptr->tab[othertab].dlink = NULL;
        otherptr->tab[othertab].dtab = 0;
    }
    *plinknum = 1 - *plinknum;
    gettabloc(tabnum,optr,tab1);
    gettabloc(othertab,otherptr,tab2);
    drawline(tab1,tab2,BLINKOFF,SOLID,ERASE);
    highlight(tabnum,*plinknum,optr,DRAW);
    return(0);
}
```

```
        /*- - - - - - - - - - - - -  - - - -  - */
        /*  D O   G S H E L L   C O M M A N D S  */
        /*- - - - - - - - - - - - - -  - - - - - */
/*
            Action: Main command loop. Repeatedly move cursor
                    while cursor keys are being pressed. If
                    another valid key is pressed, then
                    execute the command associated with this
                    key and, when done, return to main loop.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern struct onodetype *currpage;

doGshellcmds()

{
    int loc[2],nobjects,key;
    char *curswitch;
    struct onodetype *optr,*getobj();

    /* DEBUG */
    if (debug) printf("** doGshellcmds **");

    loc[X] = XMIDSCR; loc[Y] = YMIDSCR;
    do {
        cursor(loc,GIGICURSOR,DRAW);
        /* readkey switches cursor off */
        curswitch = GIGICURSORON; write(1,curswitch,strlen(curswitch));
        key = readkey();
        while (iscursor(key)){
            movecursor(loc,key,GIGICURSOR);
            key = readkey();
        }
        optr = getobj(loc,&nobjects);
        if (key == INSERT){
            if (nobjects>0) printmess("OBJECT HERE ALREADY");
            else insertobj(loc);
        }
        else if (key == EXITCHAR)
            /* do nothing */;
        else if (key==MOVE || key==WINDOW || key==DELETE || key==REFRESH ||
        key==PAGE || key==SELECT || key==EXECUTE){
            if (nobjects==0) printmess("NOTHING HERE");
            else if (nobjects>1) printmess("TOO MANY OBJECTS");
            else {
                switch (key){
                case MOVE:
                    moveobj(loc,optr); break;
```

```
                case WINDOW:
                    window(loc,optr); break;
                case DELETE:
                    delete(loc,optr); break;
                case REFRESH:
                    refresh(loc,optr); break;
                case PAGE:
                    if (OK==page(loc,optr)) currpage = optr;
                    break;
                case SELECT:
                    editlink(loc,optr); break;
                case EXECUTE:
                    execute(optr); drawpagenums(); break;
                default:
                    break;
                } /* switch */
            } /* else */
        } /* else if */
        else
            printmess("INVALID KEY");
    } while (key != EXITCHAR);
}
```

```
        /*- -  - - - - - - - - - - - - - - - - - - - */
        /*  D A T A   P A N E L   P R O C E S S    */
        /*- - -  - - - - - - - -  -  - - - - - */
/*
            Action: Read data from a pipe, write it to the file for
                    the specified data panel and, also, write data
                    in field of data panel, clipped to its boundary.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

dpprocess(optr)

struct onodetype *optr;

{
    int i,lowx,lowy,highx,highy,locx,locy,pipeend,fd,rstatus;
    char ch,*buff,*calloc();

    /* DEBUG */
    if (debug) printf("** dpprocess **");

    lowx = max(optr->x + HMARGIN, XINF);
    lowy = max(optr->y + VMARGIN, YINF);
    highx = min(optr->x + optr->wr, XSUP) - 2*CHARSEPN;
    highy = min(optr->y + optr->h, YSUP) - LINESEPN;

    /* find a non-negative file descriptor */
    for (i=0; i<TABMAX && optr->tab[i].filenum<0; i++); /* null loop */

    /* none found */
    if (i==TABMAX){
        printmess("NO PIPE TO DATA PANEL");
        exit(NOTOK);
    }

    /* found one */
    pipeend = optr->tab[i].filenum;
    /* close all file descriptors not needed */
    for (i=3; i<20; i++){
        if (i!=pipeend)
            close(i);
    }
    /* create a file for output to data panel */
    if ((fd=creat(optr->name,NORMALMODE))<0){
        printmess("CANNOT OPEN DATA PANEL FILE");
        exit(NOTOK);
    }
    /* allocate storage for an output buffer */
```

```
    buff = calloc(NAMEMAX,sizeof(char));

    /* clear data panel */
    sprintf(buff,"P[%d,%d]W(S1,E)P[,+%d]V[+%d]W(S0,V)",
    lowx,lowy,highy-lowy,highx-lowx+CHARSEPN);
    write(1,buff,strlen(buff));

    /* read from pipe, write to dpanel file, print in dpanel on screen */
    locx = lowx; locy = lowy;
    while ((rstatus=read(pipeend,&ch,1))>0){
        write (fd,&ch,1);
        if (ch==TABCHAR) ch = SPACECHAR;
        if (locy<highy){
            if (ch==NEWLINE){
                locx = lowx; locy += LINESEPN;
            }
            else if (locx<highx && ch>=SPACECHAR && ch<=TILDECHAR){
                sprintf(buff,"P[%d,%d]@IV))'%c'",locx,locy,ch);
                write(1,buff,strlen(buff));
                locx += CHARSEPN;
            }
        } /* if */
    } /* while */
    cfree(buff);
    close(pipeend); close(fd);
    exit(-rstatus);
}
```

```
        /*- - - - - - - - - - - - */
        /*  D R A W   F R A M E   */
        /*- - - - - - - - - - - - */
/*
            Action: Draw frame of page, including titles, command field
                    and page nodes (tabs down the side of the frame).

*/

#include "Gshell.defs"

extern int debug;
extern struct onodetype *currpage;

drawframe()

{
    int y,loc[2],nobjects;
    char video,*buff,*calloc();
    struct onodetype *pageptr,*getobj();

    /* DEBUG */
    if (debug) printf("** drawframe **");

    buff = calloc(48,sizeof(char));

    /* initialize screen and pen */
    sprintf(buff,"@K"); write(1,buff,strlen(buff));
    /* draw outer frame */
    sprintf(buff,"P[0,%d]@RP[0,-2]C(CA-90)[+%d]",BORDERWD+2,BORDERWD);
    write(1,buff,strlen(buff));
    sprintf(buff,"V[+%d]C(CA-90)[,+%d]",PAGEWD,BORDERWD); write(1,buff,strlen(buff));
    sprintf(buff,"@QV[,+%d]",OUTFRMHT); write(1,buff,strlen(buff));
    sprintf(buff,"@RV[,+%d]C(CA-90)[-%d]",BORDERWD,BORDERWD);
    write(1,buff,strlen(buff));
    sprintf(buff,"V[-%d]C(CA-90)[,-%d]",PAGEWD,BORDERWD); write(1,buff,strlen(buff));
    sprintf(buff,"@QV[,-%d]",OUTFRMPL16); write(1,buff,strlen(buff));

    /* draw inner frame */
    sprintf(buff,"P[%d,%d]",BORDERWD,BORDERWD2); write(1,buff,strlen(buff));
    sprintf(buff,"V[+%d]V[,+%d]",PAGEWD,PAGEHT); write(1,buff,strlen(buff));
    sprintf(buff,"V[-%d]V[,-%d]",PAGEWD,PAGEHT); write(1,buff,strlen(buff));

    /* draw page tabs */
    loc[X] = XPAGE0;
    for (y=YPAGE0; y<=YPAGE7; y += PNODESEPN){
        loc[Y] = y;
        pageptr = getobj(loc,&nobjects);
        if (nobjects != 1){
            printmess("PAGE TAB MISALIGNED");
            cfree(buff);
            return(1);
        }
```

```
        video = (pageptr==currpage) ? REVERSE : NORMAL;
        drawpagenode(pageptr,video);
    } /* for y */

    /* draw titles */
    sprintf(buff,"P[%d,0]@E'Gshell'",BORDERWD); write(1,buff,strlen(buff));
    sprintf(buff,"P[%d,0]@E[-10]'38/ka'",PAGEWD); write(1,buff,strlen(buff));
    sprintf(buff,"W(N0)"); write(1,buff,strlen(buff));

    /* restore command line */
    echokeys();

    cfree(buff);
    return(0);
}



        /*- - - - -  - -  - - - -    - */
        /*  DRAW  PAGE  NODE  */
        /*- - - - - - - - - - - - - - - */
/*
            Action: Draw page node. Number it if the corresponding page
                    is not empty or is the current page. Highlight node
                    if corresponding page is the current page.

*/

drawpagenode(pageptr,video)

struct onodetype *pageptr;
char video;

{
    int pnodew,pnodeh,tabrad;
    char *buff,*calloc();

    /* DEBUG */
    if (debug) printf("** drawpagenode **");

    buff = calloc(24,sizeof(char));

    /* print the tab */
    pnodew = pageptr->wr; pnodeh = pageptr->h; tabrad = pnodeh/2;
    sprintf(buff,"P[%d,%d]",pageptr->x,pageptr->y+tabrad); write(1,buff,strlen(buff));
    sprintf(buff,"@RC(A180)[,-%d]",tabrad); write(1,buff,strlen(buff));

    /* print the box */
    if (video == NORMAL){
        sprintf(buff,"W(S0)"); write(1,buff,strlen(buff));
    }
    sprintf(buff,"P[,+%d]V[+%d]",tabrad,pnodew); write(1,buff,strlen(buff));
    sprintf(buff,"V[,-%d]V[-%d]@Q",pnodeh,pnodew); write(1,buff,strlen(buff));
```

```
    /* print the page number */
    if (fnotempty(pageptr->name) || pageptr==currpage){
        sprintf(buff,"P[+%d,+%d]",XPNUMOFFST,YPNUMOFFST); write(1,buff,strlen(buff));
        sprintf(buff,"@H'%d'",pageptr->ord-ORDPAGE0); write(1,buff,strlen(buff));
    }
    cfree(buff);
}
```

```
        /*- - - - - - - - - - - - */
        /*  D R A W   L I N E    */
        /*- - - -   - - - - - - - */
/*
            Action: Draw line between specified points, clipped to boundary
                    of work area. Line may be dotted or solid, may blink or
                    not. Drawing includes erasure (writing style - erase).
                    Line is drawn in a consistent direction  in  order  to
                    avoid problems with the GIGI line drawing algorithm.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

drawline(loc1,loc2,blink,style,mode)

int loc1[2],loc2[2];
char blink,style,mode;

{
   int clip1[2],clip2[2];

   /* DEBUG */
   if (debug) printf("** drawline **");

   clip1[X] = loc1[X]; clip1[Y] = loc1[Y];
   clip2[X] = loc2[X]; clip2[Y] = loc2[Y];
   if (clip(clip1,clip2)){
      /* draw line in a consistent direction to avoid crumbs */
      if (clip1[X]>clip2[X] || (clip1[X]==clip2[X] && clip1[Y]>clip2[Y]))
         line(clip2,clip1,blink,style,mode);
      else
         line(clip1,clip2,blink,style,mode);
   }
   return(0);
}


        /*- - - - - - - - - - - - - */
        /*  C U R S O R   L I N E   */
        /*- - - - - - - - - - - - - */
/*
            Action: Draw special style of line to represent cursor, when
                    inserting links.

*/

cursorline(tab,loc,mode)

int tab[2],loc[2];
```

```
char mode;

{
    char blink;

    /* DEBUG */
    if (debug) printf("** cursorline **");

    blink = (mode==DRAW) ? BLINKON : BLINKOFF;
    line(tab,loc,blink,CURSLINSTYLE,COMPLMNT);
    return(0);
}


        /*- - -   - - */
        /*  L I N E   */
        /*- - - - - - */
/*
            Action: Draw line between specified points. No clipping.
                    This is the primitive routine used by all line
                    drawing procedures.

*/

line(loc1,loc2,blink,style,mode)

int loc1[2],loc2[2];
char blink,style,mode;

{
    char *buff,*calloc();

    buff = calloc(16,sizeof(char));
    sprintf(buff,"P[%d,%d]",loc1[X],loc1[Y]); write(1,buff,strlen(buff));
    sprintf(buff,"@M%c,P%c,%c))",blink,style,mode); write(1,buff,strlen(buff));
    sprintf(buff,"[%d,%d]",loc2[X],loc2[Y]); write(1,buff,strlen(buff));
    cfree(buff);
    return(0);
}
```

```
        /*- - - -   - - -   - */
        /*  D R A W   M A P   */
        /*- - - - - - - -   - */
/*
            Action: Draw all objects pointed to by that portion of the
                    page map within the limits specified by 'low' and
                    'high'.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern int pagedrawflag;
extern struct pnodetype *pagemap[YGRIDMAX][XGRIDMAX];

drawmap(low,high,penstyle)

int low[2],high[2];
char penstyle;

{
    int i,j;
    struct onodetype *optr;
    struct pnodetype *pptr;

    /* DEBUG */
    if (debug) printf("** drawmap **");

    if (penstyle == ERASE) {
        printf("S(E)"); fflush(stdout);
        return(0);
    }

    /* draw objects within field specified */
    for (i=low[X]; i<high[X]; i++)
        for (j=low[Y]; j<high[Y]; j++){
            pptr = pagemap[j][i];
            while (pptr!=NULL){
                optr = pptr->obj;
                if (optr->drawflag == pagedrawflag){
                    drawobj(optr,DRAW);
                    optr->drawflag = !pagedrawflag;
                } /* if */
                pptr = pptr->next;
            } /* while */
        } /* for j */

    /* reset remaining drawflags */
    for (i=0; i<XGRIDMAX; i++)
        for (j=0; j<YGRIDMAX; j++){
            if (i<low[X] || i>=high[X] || j<low[Y] || j>=high[Y]){
```

```
            pptr = pagemap[j][i];
            while (pptr!=NULL){
               pptr->obj->drawflag = !pagedrawflag;
               pptr = pptr->next;
            } /* while */
         } /* if */
      } /* for j */
   pagedrawflag = !pagedrawflag;
   return(0);
}
```

```
        /*- - -   - - - - - - -   -   */
        /*  D R A W   O B J E C T   */
        /*-   -     - - - -   -     */
/*
            Action: Draw specified object. Each object consists of a frame
                    and, possibly, some text.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

drawobj(optr,mode)

struct onodetype *optr;
char mode;

{
    /* DEBUG */
    if (debug) printf("** drawobj **");

    drawobjframe(optr,mode);
    if (optr->oclass != START && optr->oclass != TEE)
        drawobjtext(optr,mode);
    return(0);
}
```

```
        /*- - - - - - - - - - -  - - - - - - - - */
        /*  D R A W   O B J E C T   F R A M E   */
        /*- - - - - - - - - - - -  - - - - - - - */
/*
        Action: Draw frame of given object according to its type  and
                the dimensions given in its object-node. This includes
                the drawing of the object's tabs and  any  links
                attached to them.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

drawobjframe(optr,mode)

struct onodetype *optr;
char mode;

{
    int low[2],high[2],i,ctabnum,dtabnum;
    int tab1[2],tab2[2];
    struct onodetype *cptr,*dptr;

    /* DEBUG */
    if (debug) printf("** drawobjframe **");

    switch (optr->oclass){
    case COMMAND:
    case DPANEL:
        low[X] = optr->x; low[Y] = optr->y;
        high[X] = low[X] + optr->wr;
        high[Y] = low[Y] + optr->h;
        box(low,high,mode);
        break;
    case IIF:
    case UNTIL:
    case NOTUNTIL:
    case FFOR:
        printmess("NOT IMPLEMENTED");
        return(1);
        break;
    default:
        break;      /* no box for TEE, START, PARAM */
    } /* switch */

    /* draw tabs and links */
    for (i=0; i<TABMAX && optr->tab[i].tclass!=NOTUSED; i++){
        drawtab(i,optr,BLINKOFF,mode);
        cptr = optr->tab[i].clink;
        ctabnum = optr->tab[i].ctab;
```

```
        dptr = optr->tab[i].dlink;
        dtabnum = optr->tab[i].dtab;
        if (cptr!=NULL || dptr!=NULL){
            gettabloc(i,optr,tab1);
            if (dptr!=NULL){
                gettabloc(dtabnum,dptr,tab2);
                drawline(tab1,tab2,BLINKOFF,SOLID,mode);
            }
            if (cptr!=NULL && (cptr!=dptr || ctabnum!=dtabnum)){
                gettabloc(ctabnum,cptr,tab2);
                drawline(tab1,tab2,BLINKOFF,DOTTED,mode);
            } /* if-else */
        } /* if */
    } /* for i */
}
```

```
        /*- - - - - - - - - - - - - - - - - -  */
        /*  D R A W   O B J E C T   T E X T   */
        /*- -   - - - - - - - - - -   - - - - */
/*
          Action: Draw text associated with given object.  The  text is
                  read from  its  object-node (in the case of  COMMAND,
                  PARAM, etc.) or from an associated file ( for DPANEL).

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

drawobjtext(optr,mode)

struct onodetype *optr;
char mode;

{
    int i,locx,locy,low[2],high[2];
    char *name,*filename,*buff,*calloc();

    /* DEBUG */
    if (debug) printf("** drawobjtext **");

    switch (optr->oclass){
    case COMMAND:
        locx = optr->x + HMARGIN;
        locy = optr->y + VMARGIN;
        name = optr->name;
        if (locy<YSUP-LINESEPN){
            buff = calloc(NAMEMAX,sizeof(buff));
            sprintf(buff,"P[%d,%d]",locx,locy); write(1,buff,strlen(buff));
            for (i=0; name[i]!=NULLCHAR && locx<XSUP-CHARSEPN; i++){
                sprintf(buff,"@I"); write(1,buff,strlen(buff));
                sprintf(buff,"%c)'%c'",mode,name[i]); write(1,buff,strlen(buff));
                locx += CHARSEPN;
            }
            cfree(buff);
        }
        return(0); break;
    case PARAM:
        locx = optr->x + TABRAD;
        locy = optr->y;
        name = optr->name;
        if (locy<YSUP-LINESEPN){
            buff = calloc(NAMEMAX,sizeof(buff));
            sprintf(buff,"P[%d,%d]",locx,locy); write(1,buff,strlen(buff));
            for (i=0; name[i]!=NULLCHAR && locx<XSUP-CHARSEPN; i++){
                sprintf(buff,"@I"); write(1,buff,strlen(buff));
                sprintf(buff,"%c)'%c'",mode,name[i]); write(1,buff,strlen(buff));
```

```
            locx += CHARSEPN;
         }
         cfree(buff);
      }
      return(0); break;
   case DPANEL:
      low[X] = max(optr->x + HMARGIN, XINF);
      low[Y] = max(optr->y + VMARGIN, YINF);
      high[X] = min(optr->x + optr->wr, XSUP) - 2*CHARSEPN;
      high[Y] = min(optr->y + optr->h, YSUP) - LINESEPN;
      if (fnotempty(optr->name))
         copytxtfromf(low,high,optr->name,mode);
      return(0); break;
   case IIF: case UNTIL: case NOTUNTIL: case FFOR:
      return(1); break;
   default:
      return(1); break;
   }
}


      /*- - -  - - - - - - - - -  - -  - - -    */
      /*  C O P Y   T E X T   F R O M   F I L E  */
      /*- - - - - - - -  -    -    - - - - - - */
/*
         Action: Read text from a file and draw on (erase from) screen
                 clipped to the field specified by 'low' and 'high'.

*/


copytxtfromf(low,high,filename,mode)

int low[2],high[2];
char *filename,mode;

{
   int locx,locy,charact;
   char *buff,*calloc();
   FILE *fp,*fopen();

   /* DEBUG */
   if (debug) printf("** copytxtfromf **");

   if ((fp=fopen(filename,"r")) == NULL)
      return(0);
   else {
      locx = low[X]; locy = low[Y];
      buff = calloc(16,sizeof(char));
      sprintf(buff,"P[%d,%d]",locx,locy); write(1,buff,strlen(buff));
      while ((charact=getc(fp))!=EOF && locy<high[Y]){
         if (charact == TABCHAR) charact = SPACECHAR;
         if (charact == '0'){
```

```
            locx = low[X]; locy += LINESEPN;
            sprintf(buff,"P[%d,%d]",locx,locy); write(1,buff,strlen(buff));
        }
        else if (locx<high[X] && charact>=SPACECHAR && charact<=TILDECHAR){
            sprintf(buff,"@I"); write(1,buff,strlen(buff));
            sprintf(buff,"%c))'%c'",mode,charact); write(1,buff,strlen(buff));
            locx += CHARSEPN;
        }
    } /* while */
    cfree(buff);
    fclose(fp); return(0);
  } /* if */
}
```

```
        /*-  - - - - - - - - - - - - - -  -  */
        /*  D R A W   P A G E   N U M B E R S  */
        /*- - - - - - - - - - - - - - - - - - */
/*
            Action: Redraw page numbers in page nodes. Invoked after
                    EXECUTE, in case EXECUTE has changed the status
                    of any page.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern struct onodetype *currpage;

drawpagenums()

{
   int loc[2],y,ignore;
   char *buff,*calloc();
   struct onodetype *pageptr,*getobj();

   /* DEBUG */
   if (debug) printf("** drawpagenums **");

   buff = calloc(16,sizeof(char));

   loc[X] = XPAGE0;
   for (y=YPAGE0; y<=YPAGE7; y += PNODESEPN){
      loc[Y] = y;
      pageptr = getobj(loc,&ignore);
      sprintf(buff,"P[%d,%d]",pageptr->x+XPNUMOFFST,pageptr->y+YPNUMOFFST);
      write(1,buff,strlen(buff));
      if (pageptr==currpage){
         sprintf(buff,"@E%d",pageptr->ord-ORDPAGE0);
         write(1,buff,strlen(buff));
      }
      else if (fnotempty(pageptr->name)){
         sprintf(buff,"@F%d",pageptr->ord-ORDPAGE0);
         write(1,buff,strlen(buff));
      }
   } /* for */
   return(0);
}
```

```
        /*- - - - - - - - - - */
        /*  D R A W   T A B   */
        /*- - - -   - - - - - */
/*
            Action: Draw or erase a given tab on a specified object.
                    Tab may be made to blink.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

drawtab(tabnum,optr,blink,mode)

int tabnum;
char blink,mode;
struct onodetype *optr;

{
   int objclass,center[2],i,taboff;
   char tabclass,shade,c;

   /* DEBUG */
   if (debug) printf("** drawtab **");

   objclass = optr->oclass;
   switch (objclass){
   case COMMAND:
      tabclass = optr->tab[tabnum].tclass;
      switch (tabclass){
      case FLAGTAB:
         center[X] = optr->x + optr->wr; center[Y] = optr->y;
         sectSW(center,TABRAD,SHADEOFF,blink,mode);
         return(0); break;
      case INTAB: case INSYNCTAB:
         for (i=0; !isintab(optr->tab[i].tclass); i++) /* empty loop */ ;
         taboff = tabnum - i;
         center[X] = optr->x + taboff*TABSEPN + TABRAD;
         center[Y] = optr->y;
         shade = (tabclass==INTAB) ? SHADEON : SHADEOFF;
         sectNW(center,TABRAD,shade,blink,mode);
         sectNE(center,TABRAD,shade,blink,mode);
         return(0); break;
      case OUTTAB: case OUTSYNCTAB:
         for (i=0; !isouttab(optr->tab[i].tclass); i++) /* empty loop */ ;
         taboff = tabnum - i;
         center[X] = optr->x + taboff*TABSEPN + TABRAD;
         center[Y] = optr->y + optr->h;
         shade = (tabclass==OUTTAB) ? SHADEON : SHADEOFF;
         sectSW(center,TABRAD,shade,blink,mode);
         sectSE(center,TABRAD,shade,blink,mode);
```

```
            return(0); break;
        default:
            printmess("BAD TAB");
            return(1); break;
        }
        break;
    case START:
        if (tabnum==0 || tabnum==1){
            center[X] = optr->x; center[Y] = optr->y;
            if (tabnum == 0){
                sectNW(center,STARTRAD,SHADEOFF,blink,mode);
                sectNE(center,STARTRAD,SHADEOFF,blink,mode);
            }
            else {
                sectSW(center,STARTRAD,SHADEOFF,blink,mode);
                sectSE(center,STARTRAD,SHADEOFF,blink,mode);
            }
            return(0); break;
        }
        else {
            printmess("BAD TAB");
            return(1); break;
        }
        break;
    case TEE:
        switch (tabnum){
        case 0:
            center[X] = optr->x; center[Y] = optr->y;
            sectNW(center,TABRAD2,SHADEON,blink,mode);
            sectNE(center,TABRAD2,SHADEON,blink,mode);
            return(0); break;
        case 1:
            center[X] = optr->x - TABRAD; center[Y] = optr->y;
            sectSW(center,TABRAD,SHADEON,blink,mode);
            sectSE(center,TABRAD,SHADEON,blink,mode);
            return(0); break;
        case 2:
            center[X] = optr->x + TABRAD; center[Y] = optr->y;
            sectSW(center,TABRAD,SHADEON,blink,mode);
            sectSE(center,TABRAD,SHADEON,blink,mode);
            return(0); break;
        default:
            printmess("BAD TAB");
            return(1); break;
        }
    case PARAM:
        switch (tabnum){
        case 0:
            center[X] = optr->x + TABRAD; center[Y] = optr->y + TABRAD;
            sectNW(center,TABRAD,SHADEOFF,blink,mode);
            sectSW(center,TABRAD,SHADEOFF,blink,mode);
            return(0); break;
        case 1:
```

```
            center[X] = optr->x + optr->wr - TABRAD;
            center[Y] = optr->y + TABRAD;
            sectNE(center,TABRAD,SHADEOFF,blink,mode);
            sectSE(center,TABRAD,SHADEOFF,blink,mode);
            return(0); break;
         default:
            printmess("BAD TAB");
            return(1); break;
         }
         break;
      case DPANEL:
         switch (tabnum){
         case 0:
            center[X] = optr->x; center[Y] = optr->y;
            sectSE(center,TABRAD,SHADEON,blink,mode);
            return(0); break;
         case 1:
            center[X] = optr->x + optr->wr; center[Y] = optr->y;
            sectSW(center,TABRAD,SHADEON,blink,mode);
            return(0); break;
         case 2:
            center[X] = optr->x; center[Y] = optr->y + optr->h;
            sectNE(center,TABRAD,SHADEON,blink,mode);
            return(0); break;
         case 3:
            center[X] = optr->x + optr->wr; center[Y] = optr->y + optr->h;
            sectNW(center,TABRAD,SHADEON,blink,mode);
            return(0); break;
         default:
            printmess("BAD TAB");
            return(1); break;
         }
         break;
      case PAGE:
         center[X] = optr->x; center[Y] = optr->y + PTABRAD;
         sectsetup(center,PTABRAD,SHADEON,blink,mode);
         c = '2'; write(1,&c,1);
         sectsetup(center,PTABRAD,SHADEON,blink,mode);
         c = '4'; write(1,&c,1);
         return(0);
         break;
      case IIF:
      case UNTIL:
      case NOTUNTIL:
      case FFOR:
         printmess("NOT YET IMPLEMENTED");
         return(1);
         break;
      default:
         printmess("NO SUCH OBJECT");
         return(1);
         break;
      }

}
```

```
        /*- - - -  - - - - - - */
        /*  E C H O   K E Y S  */
        /*-  - - - - -    -  -  */
/*
            Action: Clear message field and echo last 8 keystrokes in
                    command field.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern int oldmessage;
extern int keys[NKEYS];

echokeys()

{
   int key,i;
   char *buff,*calloc();

   /* DEBUG */
   if (debug) printf("** echokeys **");

   /* allocate space for output buffer */
   buff = calloc(48,sizeof(char));

   /* clear the message field */
   if (oldmessage){
      sprintf(buff,"P[%d,%d]@RP[,-%d]V[+%d]@Q",XMESSBEGIN,YGIGIHI,
      BORDERWD2-1, MESSLEN);
      write(1,buff,strlen(buff));
      oldmessage = FALSE;
   }

   /* update key history in command field */
   sprintf(buff,"P[%d,%d]",XCMDBEGIN+CMDLEN,YCMDBEGIN);
   write(1,buff,strlen(buff));
   for (i=NKEYS-1; i>=0; i--){
      key = keys[i];
      if (key<SPACECHAR){        /* control key */
         key += ATCHAR;
         sprintf(buff,"@F[-%d,]'%c'",CHARSEPN,key);
      }
      else if (key>MAXASCII){    /* keypad & cursor keys */
         key &= MAXASCII;
         sprintf(buff,"@G[-%d,]'%c'",CHARSEPN,key);
      }
      else {                     /* ordinary keys */
         sprintf(buff,"@E[-%d,]'%c'",CHARSEPN,key);
      }
      write(1,buff,strlen(buff));
```

```
    } /* for i */
    cfree(buff);
    return(0);
}
```

```
        /*- - - - - - - - -  - */
        /*  E D I T   L I N K  */
        /*- - - - -  - - - - - */
/*
            Action: Command loop for insertion and deletion of links between
                    nodes.  Allows user to cycle  through tabs  of  current
                    object or insert or delete a link from one of its tabs.
                    The loop continues until END is pressed.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

editlink(loc,optr)

int loc[2];
struct onodetype *optr;

{
   int tabnum,linknum,key;

   /* DEBUG */
   if (debug) printf("** editlink **");

   tabnum = 0;
   if (optr->tab[tabnum].clink!=NULL)
      linknum = 0;
   else
      linknum = 1;
   highlight(tabnum,linknum,optr,DRAW);
   do {
      key = readkey();
      switch (key){
      case SELECT:
         highlight(tabnum,linknum,optr,ERASE);
         nextlink(&tabnum.&linknum,optr);
         highlight(tabnum,linknum,optr,DRAW);
         break;
      case INSERT:
         insertlink(tabnum,&linknum,optr);
         break;
      case DELETE:
         deletelink(tabnum,&linknum,optr);
         break;
      case END:
         break;
      default:
         printmess("SELECT, INSERT, DELETE, OR END");
         break;
      } /* switch */
```

```
    } while (key != END);
    highlight(tabnum,linknum,optr,ERASE);
    return(0);
}
```

```
        /*- - - - -  - - - - -  - - -*/
        /* EXECUTE  GSHELL  */    /* MAIN PROGRAM */
        /*- - - - - - - - - - -  - - -*/
/*
        Action: Calls initialization routines and selects and draws page0.
                It then calls the command processor 'doGshellcmds'.  On
                return from this it copies out the current page to the
                corresponding page file and calls the clean-up routines.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern struct onodetype *currpage;

main()

{
    int nobjects,loc[2],low[2],high[2];
    struct onodetype *getobj();

    /* DEBUG */
    debug = 0;
    if (debug) printf("** execGshell **");

    initGshell();
    initGIGI();
    loc[X] = XPAGE0; loc[Y] = YPAGE0;
    currpage = getobj(loc,&nobjects);
    if (nobjects!=1) printmess("PAGE TABS MISSALIGNED");
    drawframe();
    if (fnotempty(currpage->name))
        copypfromf(currpage->name);
    low[X] = low[Y] = 0;
    high[X] = XPAGELIM; high[Y] = YPAGELIM;
    drawmap(low,high,DRAW);
    doGshellcmds();
    copyptof(currpage->name);
    exitGshell();
    exitGIGI();
    exit(0);
}
```

```
        /*- - - - - - - - - */
        /*  E X E C U T E   */
        /*- - - - - - - - - */
/*
            Action: Executes tableau attached to selected START node.  After
                    getting the argument list attached to the START node and
                    initializing the data structures used to manage  the
                    execution process, it opens pipes for keyboard I/O and
                    forks the processes which manage this I/O.  It, then,
                    alternately executes groups of concurrent processes and
                    waits for processes to die, until all command processes
                    have compppleted. Finally, a clean-up is performed before
                    returning to the main command loop.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern oldmessage;

execute(optr)

struct onodetype *optr;

{
    int i,rdkbpid,wrtkbpid,procid,status,kbfds[2],who,whostat;
    char *calloc();
    proclisttype *ptrproclist;
    struct arglisttype *args;
    struct onodetype *datalink,*controllink,*doneoptr,*buryprocess();

    /* DEBUG */
    if (debug) printf("** execute **");

    /* ensure that next command after execute clears message field */
    oldmessage = TRUE;

    /* check that this is a START node */
    if (optr->oclass!=START){
        printmess("EXECUTE START NODE ONLY");
        return(1);
    }

    /* flush out any outstanding output */
    fflush(stdout);

    /* initialize data structures needed for execution */
    ptrproclist = (proclisttype *)calloc(1,sizeof(proclisttype));
    *ptrproclist = NULL;
    args = (struct arglisttype *)calloc(1,ARGNODESIZE);
    args->argstring[0] = NULLCHAR; args->begin[0] = 0; args->number = 0;
```

```
datalink = optr->tab[0].dlink;
if (datalink!=NULL && datalink->oclass==PARAM)
    getarguments(datalink->name,args);

/* mark all nodes UNBORN, all filenumbers NOFILE */
marknodes(optr);
optr->status = DEAD;
if ((controllink=optr->tab[1].clink)==NULL){
    printmess("NOTHING TO EXECUTE");
    return(1);
}

/* open a pipe for the write-to-keyboard process */
/* this pipe should have readfd=3 and writefd=4  */
if (pipe(kbfds)<0){
    printmess("CANNOT CREATE WRITE KB PIPE");
    abort(ptrproclist);
    return(1);
}

/* flush any buffered output before forking */
fflush(stdout);

/* fork the write-to-keyboard process */
wrtkbpid = fork();
if (wrtkbpid<0){
    printmess("CANNOT FORK WRITE KB PROCESS");
    abort(ptrproclist);
    return(1);
}
if (wrtkbpid==0)  /* child */
    wrtkbproc();

/* parent */
/* DEBUG */
if (debug) {printf("<w:%d>",wrtkbpid); fflush(stdout);}

/* initiate concurrent execution of the first group of commands */
if (initiate(controllink,ptrproclist,args)==NOTOK){
    abort(ptrproclist);
    return(1);
}

/* continue to execute commands until all are done or error occurs */
do {
    procid = wait(&status);

    /* DEBUG */
    if (debug) {printf("<ded:%d|stt:%d>",procid,status); fflush(stdout);}

    if (status==OK){
        doneoptr = buryprocess(procid,ptrproclist);
        if (doneoptr!=NULL){
```

```
                for (i=0; i<TABMAX; i++){
                    if (hasoutchan(doneoptr,i) &&
                    initiate(doneoptr->tab[i].clink,ptrproclist,args)==NOTOK){
                        abort(ptrproclist);
                        return(1);
                    } /* if */
                } /* for */
            } /* if */
        } /* if */
    } while (*ptrproclist!=NULL && (status==OK || status==KILLED));

    /* close wrt kb pipe and wait for wrtkb process to finish */
    close(3); close(4);
    while (wrtkbpid!=(who=wait(&whostat))){
        /* DEBUG */
        if (debug) {printf("<wded:%d|stat:%d>",who,whostat); fflush(stdout);}
    }
    /* DEBUG */
    if (debug) {printf("<wded:%d|stat:%d>",who,whostat); fflush(stdout);}


    /* if error occurs, clean up the mess */
    if (status!=OK){
        printmess("ERROR: ABNORMAL EXIT");
        abort(ptrproclist);
        return(1);
    }
    else {
        return(0);
    }
}
```

```
        /*- - - - - - - -  - - - -  - - - - - - - - - - - - - - - - */
        /* E X E C U T E  G R O U P  O F  C O N C U R R E N T  */
        /*                P R O C E S S E S                   */
        /*- - - - - - - - - -  - - - - - - -  - - -  - -  - - - - */
/*
          Action: Allocate file descriptors to all tabs of the specified
                  object which require them. Read and interpret parameters
                  from the flag tab, if there are any.  Fork the process
                  for this object and add the process to the process list.
                  Close file descriptors assigned to the process. Then,
                  for each unborn process attached to the current node do
                  the same thing. Fork processes to handle output to data
                  panels, if required.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

executegroup (optr,ptrproclist,args,pnoreaders)

struct onodetype *optr;
proclisttype *ptrproclist;
struct arglisttype *args;
int *pnoreaders;

{
   int i,procid;
   char *params,*calloc();
   struct onodetype *datalink;
   struct procnodetype *tidyup();

   /* DEBUG */
   if (debug) printf("** executegroup **");

   /* is this a legal object? */
   if (isdataobj(optr->oclass) || optr->oclass==START){
      printmess("CANNOT EXECUTE DATA OR START NODE");
      return(NOTOK);
   }

   /* are we ready to start? */
   for (i=0; i<TABMAX; i++){
      if (hasinchan(optr,i) && optr->tab[i].clink->status!=DEAD)
         return(OK);  /* not yet */
   }

   /* yes, get each tab ready */
   params = calloc(NAMEMAX,sizeof(char));
   params[0] = NULLCHAR;
   for (i=0; i<TABMAX; i++){
```

```
    switch(optr->tab[i].tclass){
    case INTAB:
        if (prepintab(optr,i,pnoreaders)==NOTOK)
            return(NOTOK);
        break;
    case OUTTAB:
        if (prepouttab(optr,i)==NOTOK)
            return(NOTOK);
        break;
    case FLAGTAB:
        if (optr->tab[i].dlink!=NULL){
            if (getparams(optr->tab[i].dlink->name,params,args)==NOTOK)
                return(NOTOK);
        }
        break;
    default:
        break;
    } /* switch */
} /* for */

/* fork the process for this command */
optr->status = ALIVE;
procid = fork();
if (procid<0){
    printmess2("CANNOT FORK ",optr->name);
    return(NOTOK);
}
if (procid==0) {   /* child */
    close(0);
    executenode(optr,params);
}

/* parent */

/* DEBUG */
if (debug) { printf("<cm:%d>",procid); fflush(stdout);}

/* add process to process list and close unwanted file descriptors */
tidyup(optr,procid,ptrproclist);

/* start processes for any commands connected by pipes and yet UNBORN */
/* also start processes for data panels connected to this command node */
for (i=0; i<TABMAX; i++){
    datalink = optr->tab[i].dlink;
    if (haspipe(optr,i)){
        if (datalink->status==DEAD){
            printmess("PIPE CONNECTED TO DEAD OBJECT");
            return(NOTOK);
        }
        if (datalink->status==UNBORN &&
        executegroup(datalink,ptrproclist,args,pnoreaders)==NOTOK)
            return(NOTOK);
    } /* if */
```

```
        if (optr->tab[i].tclass==OUTTAB && datalink->oclass==DPANEL){
            procid = fork();
            if (procid<0){
                printmess("CANNOT FORK DPANEL PROCESS");
                return(NOTOK);
            }
            if (procid==0) {   /* child */
                close(0);
                dpprocess(datalink);
            }

            /* parent */

            /* DEBUG */
            if (debug) {printf("<dp:%d>",procid); fflush(stdout);}

            tidyup(datalink,procid,ptrproclist);
        } /* if */
    } /* for */
    return(OK);
}
```

```
        /*- - - - - - - - - - - - - - - - */
        /*  E X E C U T E   N O D E   */
        /*- - - - - - - - - - - - - - - - */
/*
        Action: Invoke the UNIX command associated with the given node
                if it is a COMMAND node.  If it is a TEE, then perform
                the appropriate data transfer.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

executenode(optr,params)

struct onodetype *optr;
char *params;

{
   int infd,out1fd,out2fd,rstatus;
   char ch,*cmd,*calloc();

   /* DEBUG */
   if (debug) {printf("** executenode **"); fflush(stdout);}

   /* reassign file descriptors to in and out tabs */
   /* to provide default values */
   reassignfds(optr);

   /* execute node according to its type */
   switch (optr->oclass){
   case COMMAND:
      /* build command string */
      cmd = calloc(NAMEMAX,sizeof(char));
      strcpy(cmd,optr->name);
      strcat(cmd,params);
      /* execute command */
      execl("/bin/sh","sh","-c",cmd,0);
      printmess2("EXECUTION FAILED FOR",optr->name);
      exit(NOTOK);
      break;
   case TEE:
      infd = optr->tab[0].filenum;
      out1fd = optr->tab[1].filenum;
      out2fd = optr->tab[2].filenum;
      rstatus = 0;
      if (infd>=0){
         while ((rstatus=read(infd,&ch,1))>0){
            if (out1fd>=0)
               write(out1fd,&ch,1);
            if (out2fd>=0)
```

```
                write(out2fd,&ch,1);
            }
            close(infd);
        }
        if (out1fd>=0)
            close(out1fd);
        if (out2fd>=0)
            close(out2fd);
        exit(-rstatus);
        break;
    default:
        printmess(optr->name," NOT EXECUTABLE");
        exit(NOTOK);
        break;
    } /* switch */
}
```

```
        /*- - - - - - -   - - - */
        /*  E X I T   G I  G I   */
        /*- - - - - - - - -   - */
/*
            Action: Switch on GIGI text and graphics cursors and return
                    GIGI to text mode.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

exitGIGI()

{
   char *buff,*calloc();

   /* DEBUG */
   if (debug) printf("** exitGIGI **");

   buff = calloc(32,sizeof(char));
   /* both graphics and text cursors on */
   sprintf(buff,"\033PrVC3 33\\"); write(1,buff,strlen(buff));

   /* return to text mode */
   sprintf(buff,"\033\\"); write(1,buff,strlen(buff));
   cfree(buff);
   return(0);
}
```

```
        /*- - -  - - - - - -  - -  */
        /*  E X I T  GS H E L L  */
        /*- - - - - - - - - -  - - */
/*
        Action: Clear page map and erase screen. Switch on terminal
                echo and return to line-by-line input (as opposed to
                character-by-character or 'raw' mode).

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

exitGshell()

{
   int low[2],high[2];

   /* DEBUG */
   if (debug) printf("** exitGshell **");

   low[X] = low[Y] = 0; high[X] = XPAGELIM; high[Y] = YPAGELIM;
   drawmap(low,high,ERASE);
   clearmap();
   system("stty echo -cbreak");
}
```

```
        /*- - - - - - - - - - - - -  - - */
        /*  F I L E   N O T   E M P T Y  */
        /*- -  - - - - - - -  -   -  -  */
/*
            Action: Return TRUE if specified file exists and is not empty.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

fnotempty(fname)

char *fname;

{
   FILE *fp,*fopen();
   int status;

   /* DEBUG */
   if (debug) printf("** fnotempty **");

   if ((fp=fopen(fname,"r"))==NULL) status = FALSE;
   else if (getc(fp)==EOF) status = FALSE;
   else status = TRUE;
   if (fp!=NULL) fclose(fp);
   return(status);
}


        /*- - - - - - - - - - - -  */
        /*  F I L E   E X I S T S  */
        /*- - - - - - - -  - - - - */
/*
            Action: Returns TRUE if specified file exists.

*/

fexists(filename)

char filename[ ];

{
   FILE *fp,*fopen();

   if ((fp=fopen(filename,"r")) == NULL)
      return(FALSE);
   else {
      fclose(fp);
      return(TRUE);
   }
```

}

```
        /*- - - - - - - - - - - - - - - - - */
        /*  G E T   A R G U M E N T S      */
        /*- - - - - - - - - - - - - - -  - */
/*
            Action: Get parameter string from PARAM node attached to
                    START node and break it into separate arguments.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

getarguments(name,args)

char *name;
struct arglisttype *args;

{
   int i,j,k;

   /* DEBUG */
   if (debug) printf("** getarguments **");

   i = 0; j = 0; k = 0;
   while (name[i]!=NULLCHAR && k<NARGS-1){
      while (iswhitespace(name[i])) i++;
      if (name[i]!=NULLCHAR)
         args->begin[k++] = j;
      while (!iswhitespace(name[i]) && name[i]!=NULLCHAR)
         args->argstring[j++] = name[i++];
      if (name[i]!=NULLCHAR)
         args->argstring[j++] = SPACECHAR;
   }
   args->argstring[j] = NULLCHAR;
   args->begin[k] = j;
   args->number = k;
   return(0);
}
```

```
        /*- - - - - - - - - - - - - - - -  - -  */
        /*  G E T   E X T E N T   O B J E C T  */
        /*- - - - - - - - - -  - - - - - - - -  */
/*
        Action: Returns low and high limits of those grid locations in
                the page map whose linked lists contain a pointer to
                the given object. Note that 'low' and 'high' return
                index limits of components of page map, not actual
                screen locations.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

getextentobj(optr,low,high)

int low[2],high[2];
struct onodetype *optr;

{
   int x,y,wr,h,class;

   /* DEBUG */
   if (debug) printf("** getextentobj **");

   x = optr->x; y = optr->y;
   wr = optr->wr; h = optr->h;
   class = optr->oclass;
   if (class==START || class==TEE){
      low[X] = x-wr; high[X] = x+wr;
      low[Y] = y-wr; high[Y] = y+wr;
   }
   else {
      low[X] = x; high[X] = x+wr;
      low[Y] = y; high[Y] = y+h;
   }
   low[X] /= XMESH; high[X] /= XMESH;
   low[Y] /= YMESH; high[Y] /= YMESH;

   /* keep away from pagetabs and command line */

   high[X] = min(high[X],XPAGELIM-1);
   high[Y] = min(high[Y],YPAGELIM-1);
   return(0);
}
```

```
        /*- - - - - - - - - - */
        /*  G E T   N A M E   */
        /*- - - - - - - - - - */
/*
        Action: Read a character string from the keyboard and echo it in
                the message field.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern int oldmessage;

getname(name)

char name[NAMEMAX];

{
    int loc[2],i,key;
    char *buff,*calloc();

    /* DEBUG */
    if (debug) printf("** getname **");

    buff = calloc(32,sizeof(char));

    /* clear message field, if necessary */
    if (oldmessage){
        sprintf(buff,"P[%d,%d]@R",XMESSBEGIN,YGIGIHI);
        write(1,buff,strlen(buff));
        sprintf(buff,"P[,-%d]V[+%d]",BORDERWD2-1,MESSLEN);
        write(1,buff,strlen(buff));
    }
    sprintf(buff,"@Q"); write(1,buff,strlen(buff));

    loc[X] = XMESSBEGIN; loc[Y] = YMESSBEGIN;
    i = 0;
    while ((key=readkey()) != END){
        if (SPACECHAR<=key && key<=TILDECHAR){
            if (i<NAMEMAX){
                sprintf(buff,"P[%d,%d]@E'%c'",loc[X],loc[Y],key);
                write(1,buff,strlen(buff));
                name[i++] = key;
                loc[X] += CHARSEPN;
            } /* if */
            else
                printmess("CHAR. STRING TOO LONG");
        } /* if */
        else if (key==BACKSPACE && i>0){
            loc[X] -= CHARSEPN;
            i--;
```

```
            sprintf(buff,"P[%d,%d]@E'%c'",loc[X],loc[Y],SPACECHAR);
            write(1,buff,strlen(buff));
        } /* else if */
    } /* while */
    name[i] = ' ';
    oldmessage = TRUE;
    cfree(buff);
    return(0);
}
```

```
        /*- - - - - - - - - - */
        /*  GET  NODE  */
        /*- - -  - - - -  - */
/*
        Action: Allocate memory for an object node. Assign the name,
                class, and location of the object to the node and
                initialize each tab to the appropriate tab class and
                set all links to NULL.

*/

#include "Gshell.defs"
#define FOUND  0

        /* globals */
extern int debug;
extern struct onodetype *currpage;

struct onodetype *getnode(class,name,loc)

int class,loc[2];
char name[NAMEMAX];

{
    int i,intabs,outtabs,tabsalloc,namealloc;
    char t,cmdname[NAMEMAX],cmdtabs[TABMAX],*calloc();
    FILE *fp,*fopen();
    struct onodetype *optr;

    /* DEBUG */
    if (debug) printf("** getnode **");

    if ((optr=(struct onodetype *)calloc(1,ONODESIZE)) == NULL){
        printmess("NO MEM. SPACE FOR NODE");
        return(NULL);
    }
    optr->oclass = class;
    strcpy(optr->name,name);
    optr->x = loc[X]; optr->y = loc[Y];
    for (i=0; i<TABMAX; i++){
        optr->tab[i].tclass = NOTUSED;
        optr->tab[i].clink = optr->tab[i].dlink = NULL;
        optr->tab[i].ctab = optr->tab[i].dtab = 0;
    }

    /* handle individual object types */
    switch (class){
    case COMMAND:
        if ((fp=fopen("../Gfiles/Gshell.cmds","r")) == NULL){
            cfree(optr);
            return(NULL);
        }
        while (fscanf(fp,"%s %s",cmdname,cmdtabs)!=EOF &&
```

```
        strcmp(cmdname,name)!=FOUND)
      /* empty loop */;
   fclose(fp);
   if (strcmp(cmdname,name)!=FOUND){
      printmess2(name," NOT FOUND");
      cfree(optr);
      return(NULL);
   }
   /* object tabs */
   intabs = outtabs = 0;
   for (i=0; (t=cmdtabs[i])!=' '; i++){
      optr->tab[i].tclass = t;
      switch (t){
      case INTAB: case INSYNCTAB:
         intabs++; break;
      case OUTTAB: case OUTSYNCTAB:
         outtabs++; break;
      case FLAGTAB:
         break;
      default:
         printmess2("BAD TAB FOR ",name);
         cfree(optr);
         return(NULL);
      } /* switch */
   } /* for i */

   /* object dimensions */
   namealloc = HMARGIN2 + strlen(name)*CHARSEPN;
   tabsalloc = max(intabs,outtabs)*TABSEPN;
   optr->wr = max(namealloc,tabsalloc);
   optr->h = BOXHT;
   break;
case PARAM:
   optr->tab[0].tclass = optr->tab[1].tclass = DATATAB;
   optr->wr = TABRAD2 + strlen(name)*CHARSEPN;
   optr->h = TABRAD2;
   break;
case START:
   optr->tab[0].tclass = FLAGTAB;
   optr->tab[1].tclass = OUTSYNCTAB;
   optr->wr = STARTRAD;                    /* radius */
   optr->h = 0;                            /* therefore height not needed */
   break;
case TEE:
   optr->tab[0].tclass = INTAB;
   optr->tab[1].tclass = optr->tab[2].tclass = OUTTAB;
   optr->wr = TABRAD2;                     /* radius */
   optr->h = 0;                            /* therefore height not needed */
   break;
case DPANEL:
   sprintf(optr->name,"%s.%d.%d",currpage->name,optr->x,optr->y);
   optr->tab[0].tclass = optr->tab[1].tclass =
      optr->tab[2].tclass = optr->tab[3].tclass = DATATAB;
```

```
        optr->wr = WSTDPANEL;
        optr->h = HSTDPANEL;
        break;
    case IIF: case UNTIL: case NOTUNTIL: case FFOR:
        cfree(optr);
        return(NULL);
    default:
        cfree(optr);
        return(NULL);
    } /* switch */
}
```

```
        /*- - - - - - - - - - - - - */
        /*  G E T   O B J E C T    */
        /*- - - -   - - - - - - - - */
/*
          Action: Return pointer to object node of object which contains
                  the point 'loc'. If there is more than one such object
                  the last one found is returned. The parameter 'pnum'
                  returns the number of objects which contain 'loc'.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern struct pnodetype *pagemap[YGRIDMAX][XGRIDMAX];

struct onodetype *getobj(loc,pnum)

int loc[2],*pnum;

{
   int xgrid,ygrid,x,y,wr,h,offx,offy,inobj;
   int class;
   struct onodetype *optr,*pobj;
   struct pnodetype *pptr;

   /* DEBUG */
   if (debug) printf("** getobj **");

   optr = NULL; *pnum = 0;
   xgrid = loc[X]/XMESH; ygrid = loc[Y]/YMESH;
   pptr = pagemap[ygrid][xgrid];
   while (pptr!=NULL){
      pobj = pptr->obj;
      x = pobj->x; y = pobj->y;
      wr = pobj->wr; h = pobj->h;
      offx = loc[X]-x; offy = loc[Y]-y;
      class = pobj->oclass;
      if (class==START || class==TEE)
         inobj = (offx*offx + offy*offy <= wr*wr);
      else
         inobj = (0<=offx && offx<=wr && 0<=offy && offy<=h);
      if (inobj){
         optr = pobj; (*pnum)++;
      }
      pptr = pptr->next;
   } /* while */
   return(optr);
}
```

```
        /*- - - - -  - - - - - - - - - - - */
        /*  G E T   P A R A M E T E R S   */
        /*- - - - - - - - - - - -  - - -  */
/*
        Action: Words, separated by whitespace, are copied from 'string'
                to 'params'. Those of the form $n, where n is an integer,
                cause the nth arg. in 'args' to be copied into 'params'
                instead. The word $ causes all of 'args' to be copied
                into 'params'.

*/

#include "Gshell.defs"
#define SKIPSPACE       0
#define COPYCHAR        1
#define DOLLAR          2
#define GETINT          3

        /* globals */
extern int debug;

getparams(string,params,args)

char *string,*params;
struct arglisttype *args;

{
    int i,j,k,state,argindex;

    /* DEBUG */
    if (debug) printf("** getparams **");

    k = strlen(string);
    if (!iswhitespace(string[k-1])){
        string[k] = SPACECHAR;
        string[k+1] = NULLCHAR;
    }
    params[0] = SPACECHAR;
    i = 0; j = 1;
    state = SKIPSPACE;
    while (string[i]!=NULLCHAR && j<NAMEMAX-1){
        switch (state){
        case SKIPSPACE:
            if (iswhitespace(string[i])){
                i++; state = SKIPSPACE;
            }
            else if (string[i]=='$'){
                i++; state = DOLLAR;
            }
            else {
                params[j++] = string[i];
                i++; state = COPYCHAR;
            }
```

```
    break;
case COPYCHAR:
    if (iswhitespace(string[i])){
        params[j++] = SPACECHAR;
        i++; state = SKIPSPACE;
    }
    else {
        params[j++] = string[i];
        i++; state = COPYCHAR;
    }
    break;
case DOLLAR:
    if (isdigit(string[i])){
        argindex = string[i]-'0';
        i++; state = GETINT;
    }
    else if (iswhitespace(string[i])){
        for (k=0; args->argstring[k]!=NULLCHAR && j<NAMEMAX-1; k++)
            params[j++] = args->argstring[k];
        if (j<NAMEMAX-1){
            params[j++] = SPACECHAR;
            i++; state = SKIPSPACE;
        }
    }
    else {
        params[j++] = '$';
        if (j<NAMEMAX-1){
            params[j++] = string[i];
            i++; state = COPYCHAR;
        }
    }
    break;
case GETINT:
    if (isdigit(string[i])){
        argindex = argindex*10+string[i]-'0';
        i++; state = GETINT;
    }
    else {
        if (0<argindex && argindex<=args->number){
            for (k=args->begin[argindex-1]; k<args->begin[argindex] &&
            j<NAMEMAX; k++)
                params[j++] = args->argstring[k];
        }
        if (j<NAMEMAX-1)
            params[j++] = SPACECHAR;
        if (iswhitespace(string[i])){
            i++; state = SKIPSPACE;
        }
        else if (string[i]=='$'){
            i++; state = DOLLAR;
        }
        else if (j<NAMEMAX-1){
            params[j++] = string[i];
```

```
                    i++; state = COPYCHAR;
                }
            }
            break;
        default:
            break;
        } /* switch */
    } /* while */
    params[j] = NULLCHAR;
    return(0);
}
```

```
        /*- - - - - - - - - - - - - - -   - - - */
        /*  G E T   T A B   L O C A T I O N  */
        /*- -   - - - - - - - - - - - - - - - - */
/*
            Action: Return screen location of given tab for the purposes
                    of attaching a link or cursor line to the tab.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

gettabloc(tabnum,optr,tab)

int tabnum,tab[2];
struct onodetype *optr;

{
   int i,taboff,objclass;
   char tabclass;

   /* DEBUG */
   if (debug) printf("** gettabloc **");

   objclass = optr->oclass;
   tabclass = optr->tab[tabnum].tclass;
   switch (objclass){
   case COMMAND:
      switch (tabclass){
      case FLAGTAB:
         tab[X] = optr->x + optr->wr; tab[Y] = optr->y;
         return(0); break;
      case INTAB: case INSYNCTAB:
         for (i=0; !isintab(optr->tab[i].tclass); i++) /* empty loop */ ;
         taboff = tabnum - i;
         tab[X] = optr->x + taboff*TABSEPN + TABRAD;
         tab[Y] = optr->y - TABRAD;
         return(0); break;
      case OUTTAB: case OUTSYNCTAB:
         for (i=0; !isouttab(optr->tab[i].tclass); i++) /* empty loop */ ;
         taboff = tabnum - i;
         tab[X] = optr->x + taboff*TABSEPN + TABRAD;
         tab[Y] = optr->y + optr->h + TABRAD;
         return(0); break;
      default:
        printmess("BAD TAB");
        return(1); break;
      }
      break;
   case START:
      if (tabnum==0 || tabnum==1){
```

```
        tab[X] = optr->x; tab[Y] = optr->y + STARTRAD*tabnum;
        return(0);
    }
    else {
        printmess("BAD TAB");
        return(1);
    }
    break;
case TEE:
    switch (tabnum){
    case 0:
        tab[X] = optr->x; tab[Y] = optr->y   TABRAD2;
        return(0); break;
    case 1:
        tab[X] = optr->x - TABRAD; tab[Y] = optr->y + TABRAD;
        return(0); break;
    case 2:
        tab[X] = optr->x + TABRAD; tab[Y] = optr->y + TABRAD;
        return(0); break;
    default:
        printmess("BAD TAB");
        return(1); break;
    }
    break;
case PARAM:
    if (tabnum==0 || tabnum==1){
        tab[X] = optr->x + optr->wr * tabnum;
        tab[Y] = optr->y + TABRAD;
        return(0); break;
    }
    else {
        printmess("BAD TAB");
        return(1); break;
    }
    break;
case DPANEL:
    if (0<=tabnum && tabnum<=3){
        tab[X] = optr->x + optr->wr * (tabnum%2);
        tab[Y] = optr->y + optr->h * (tabnum/2);
        return(0); break;
    }
    else {
        printmess("BAD TAB");
        return(1); break;
    }
    break;
case PAGE:
    tab[X] = optr->x - PTABRAD; tab[Y] = optr->y + PTABRAD;
    return(0);
    break;
case IIF:
case UNTIL:
case NOTUNTIL:
```

```
    case FFOR:
        printmess("NOT YET IMPLEMENTED");
        return(1);
        break;
    default:
        printmess("NO SUCH OBJECT");
        return(1);
        break;
    }
}
```

```
        /*- - - - - - - - - - - */
        /*  H I  G H L I  G H T  */
        /*-   - - - - - - - - - */
/*
            Action: Highlight selected tab and link by drawing in BLINK
                    mode.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

highlight(tabnum,linknum,optr,mode)

int tabnum,linknum;
struct onodetype *optr;
char mode;

{
   int tab1[2],tab2[2];
   char linestyle,blink;
   struct onodetype *otheroptr;

   /* DEBUG */
   if (debug) printf("** highlight **");

   blink = (mode==DRAW) ? BLINKON : BLINKOFF;
   drawtab(tabnum,optr,blink,DRAW);
   if (linknum==0 && (otheroptr=optr->tab[tabnum].clink)!=NULL){
      gettabloc(tabnum,optr,tab1);
      gettabloc(optr->tab[tabnum].ctab,otheroptr,tab2);
      linestyle = (otheroptr==optr->tab[tabnum].dlink) ? SOLID : DOTTED;
      drawline(tab1,tab2,blink,linestyle,DRAW);
   }
   else if (linknum==1 && (otheroptr=optr->tab[tabnum].dlink)!=NULL){
      gettabloc(tabnum,optr,tab1);
      gettabloc(optr->tab[tabnum].dtab,otheroptr,tab2);
      drawline(tab1,tab2,blink,SOLID,DRAW);
   }
   return(0);
}
```

```
        /*- - - - - -  - - - - - -  - - */
        /*  I N I T I A L I Z E  G I G I  */
        /*- - - - - - - - - -  - - - - - - */
/*
            Action: Perform GIGI setups. Initialize screen and writing
                    attributes. Load GIGI's alternate alphabets with
                    with special symbols.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

initGIGI()

{
    char *buff,*calloc();

    /* DEBUG */
    if (debug) printf("** initGIGI **");

    /* allocate space for output buffer */
    buff = calloc(64,sizeof(buff));

    /* GIGI setups */
    sprintf(buff,"\33[?2h"); write(1,buff,strlen(buff));
                /* ANSI terminal mode */
    sprintf(buff,"\33[?23l"); write(1,buff,strlen(buff));
                /* Programmed keypad off */
    sprintf(buff,"\33="); write(1,buff,strlen(buff));
                /* Keypad in application mode */
    sprintf(buff,"\33[?1h"); write(1,buff,strlen(buff));
                /* Cursor keys in appl. mode */
    sprintf(buff,"\33PrVC1\33\\"); write(1,buff,strlen(buff));
                /* Graphics cursor off */
    sprintf(buff,"\33Pp"); write(1,buff,strlen(buff));
                /* GIGI graphics on */

    /* DEBUG */
    if (debug) return(0);

    /* Screen attributes: */
    /*    erase screen, black screen, normal video (not rev.) */
    sprintf(buff,"S(E,I0,N0)"); write(1,buff,strlen(buff));

    /* Writing attributes: */
    /*    white pencolor, blink off, shading off, negative writing off, */
    /*    solid pen pattern, overlay writing, movement in one pixel steps */
    sprintf(buff,"W(I7,A0,S0,N0,P1,V,M1)"); write(1,buff,strlen(buff));

    /* Load alternate alphabets */
```

```
/* Cursors */
sprintf(buff,"L(a1)"); write(1,buff,strlen(buff));

sprintf(buff,"L'A'3F0305091121C0400000"); write(1,buff,strlen(buff));
                /* ARROW */
sprintf(buff,"L'B'000808080087F08080808"); write(1,buff,strlen(buff));
                /* CROSS */
sprintf(buff,"L'C'0101010101010101FF"); write(1,buff,strlen(buff));
                /* ANGLE */
sprintf(buff,"L'D'77777777777777777777"); write(1,buff,strlen(buff));
                /* Block */

/* Keypad symbols */
sprintf(buff,"L(a2)"); write(1,buff,strlen(buff));

sprintf(buff,"L'u'7840404078111B151111"); write(1,buff,strlen(buff));
                /* COMMAND */
sprintf(buff,"L's'20202020200F080E0808"); write(1,buff,strlen(buff));
                /* IIF */
sprintf(buff,"L'q'0102020408081020204O"); write(1,buff,strlen(buff));
                /* PIPE */
sprintf(buff,"L'r'0020200008080002020O"); write(1,buff,strlen(buff));
                /* CHANNEL */
sprintf(buff,"L'n'78407040400F090F0A09"); write(1,buff,strlen(buff));
                /* FFOR */
sprintf(buff,"L't'7C101010100F080E080F"); write(1,buff,strlen(buff));
                /* TEE */
sprintf(buff,"L'm'78407808781F04040404"); write(1,buff,strlen(buff));
                /* START */
sprintf(buff,"L'v'70484848700F090F0808"); write(1,buff,strlen(buff));
                /* DPANEL */
sprintf(buff,"L'l'78487840400F090F0A09"); write(1,buff,strlen(buff));
                /* PARAM */
sprintf(buff,"L'Q'446C54444411110A0A04"); write(1,buff,strlen(buff));
                /* MOVE */
sprintf(buff,"L'P'4040404040119151311"); write(1,buff,strlen(buff));
                /* INSERT */
sprintf(buff,"L'R'4444546C441119151311"); write(1,buff,strlen(buff));
                /* WINDOW */
sprintf(buff,"L'S'704848487008080808OF"); write(1,buff,strlen(buff));
                /* DELETE */
sprintf(buff,"L'w'78487850480F080E0808"); write(1,buff,strlen(buff));
                /* REFRESH */
sprintf(buff,"L'y'78487840400F080B090F"); write(1,buff,strlen(buff));
                /* PAGE */
sprintf(buff,"L'x'784078087808080808OF"); write(1,buff,strlen(buff));
                /* SELECT */
sprintf(buff,"L'M'7840704078110A040A11"); write(1,buff,strlen(buff));
                /* EXECUTE */
sprintf(buff,"L'p'78407040781119151311"); write(1,buff,strlen(buff));
                /* END */

/* Cursor key symbols */
```

```
sprintf(buff,"L'A'00081C2A490808080800"); write(1,buff,strlen(buff));
              /* UPCURSOR */
sprintf(buff,"L'B'0008080808492A1C0800"); write(1,buff,strlen(buff));
              /* DOWNCURSOR */
sprintf(buff,"L'C'000804027F0204080000"); write(1,buff,strlen(buff));
              /* RIGHTCURSOR */
sprintf(buff,"L'D'000810207F2010080000"); write(1,buff,strlen(buff));
              /* LEFTCURSOR */

/* macrographs */

sprintf(buff,"@:A T(S[16,20]M[2,2]A1)'A' @;"); write(1,buff,strlen(buff));
              /* ARROW CURSOR */
sprintf(buff,"@:B T(S[16,20]M[2,2]A1)'B' @;"); write(1,buff,strlen(buff));
              /* CROSS CURSOR */
sprintf(buff,"@:C T(S[16,20]M[2,2]A1)'C' @;"); write(1,buff,strlen(buff));
              /* ANGLE CURSOR */
sprintf(buff,"@:D W(N1)T(S1,H2,A1)'D'W(N0) @;"); write(1,buff,strlen(buff));
              /* BLOCK CURSOR */
sprintf(buff,"@:E T(S1,A0,W(N1,R)) @;"); write(1,buff,strlen(buff));
              /* CHAR. SET 0, NEG. WRITING */
sprintf(buff,"@:F T(S1,A0,W(N0,R)) @;"); write(1,buff,strlen(buff));
              /* CHAR. SET 0, POS. WRITING */
sprintf(buff,"@:G T(S1,A2,W(N1,R)) @;"); write(1,buff,strlen(buff));
              /* CHAR. SET 2, NEG. WRITING */
sprintf(buff,"@:H T(S1,H2,A0,W(C)) @;"); write(1,buff,strlen(buff));
              /* CHAR. SET 0, STAND. SIZE */
sprintf(buff,"@:I T(S1,H2,A0,W(A0,S0,N0,P1, @;"); write(1,buff,strlen(buff));
              /* CHR. SET 0, STND. SIZE, NO BLINK OR SHADE, POS. WRIT.*/
sprintf(buff,"@:J V(W(A0,S0,N0,P1, @;"); write(1,buff,strlen(buff));
              /* DRAW LINE, NO BLINK OR SHADE, POS. WRIT., SOLID PEN */
sprintf(buff,"@:K S(I0,N0)W(I7,A0,N0,P1,V,M1) @;"); write(1,buff,strlen(buff));
              /* SET SCREEN & PEN ATTRIBUTES */
sprintf(buff,"@:L @;"); write(1,buff,strlen(buff));
              /* NOT USED */
sprintf(buff,"@:M V(W(S0,N0,A @;"); write(1,buff,strlen(buff));
              /* DRAW LINE, NO SHADE, POS. WRIT. */
sprintf(buff,"@:N W(A1,C) @;"); write(1,buff,strlen(buff));
              /* BLINK ON, COMPLEMENT WRITING */
sprintf(buff,"@:O W(A0,C) @;"); write(1,buff,strlen(buff));
              /* BLINK OFF, COMPLEMENT WRITING */
sprintf(buff,"@:P W(A0,V) @;"); write(1,buff,strlen(buff));
              /* BLINK OFF, OVERLAY WRITING */
sprintf(buff,"@:Q W(S0) @;"); write(1,buff,strlen(buff));
              /* SHADING OFF */
sprintf(buff,"@:R W(S1) @;"); write(1,buff,strlen(buff));
              /* SHADING ON */

/* macrographs - setups for drawing tab sectors */

sprintf(buff,"@:S C(W(S0,A0,E,M @;"); write(1,buff,strlen(buff));
              /* SHADE OFF, BLINK OFF, ERASE */
sprintf(buff,"@:T C(W(S0,A0,V,M @;"); write(1,buff,strlen(buff));
```

```
                    /* SHADE OFF, BLINK OFF, OVERLAY */
    sprintf(buff,"@:U C(W(S0,A1,E,M @;"); write(1,buff,strlen(buff));
                    /* SHADE OFF, BLINK ON, ERASE */
    sprintf(buff,"@:V C(W(S0,A1,V,M @;"); write(1,buff,strlen(buff));
                    /* SHADE OFF, BLINK ON, OVERLAY */
    sprintf(buff,"@:W C(W(S1,A0,E,M @;"); write(1,buff,strlen(buff));
                    /* SHADE ON, BLINK OFF, ERASE */
    sprintf(buff,"@:X C(W(S1,A0,V,M @;"); write(1,buff,strlen(buff));
                    /* SHADE ON, BLINK OFF, OVERLAY */
    sprintf(buff,"@:Y C(W(S1,A1,E,M @;"); write(1,buff,strlen(buff));
                    /* SHADE ON, BLINK ON, ERASE */
    sprintf(buff,"@:Z C(W(S1,A1,V,M @;"); write(1,buff,strlen(buff));
                    /* SHADE ON, BLINK ON, OVERLAY */

    cfree(buff);
    return(0);
}
```

```
        /*- - - - - - - - - - - - - - - - - - - - - */
        /*  I N I T I A L I Z E   G S H E L L   */
        /*- - - - - - - - - - -   - - - - - - - - - */
/*
            Action: Disable echo and put system in 'raw' mode. Initialize
                    key-stroke history array and page map. Allocate page
                    nodes and insert pointers to them in page map

*/

#include "Gshell.defs"

        /* globals */
int debug = 0;
int pagedrawflag = 0;
int oldmessage = 0;
int keys[NKEYS];
struct onodetype *currpage;
struct pnodetype *pagemap[YGRIDMAX][XGRIDMAX];

initGshell()

{
   int i,j,status;
   char *calloc();
   struct onodetype *optr;
   struct pnodetype *pptr;

   /* DEBUG */
   if (debug) printf("** initGshell **");

   /* disable echo and put stdin in 'raw' mode */
   /* that is, make characters available as */
   /* received, do not wait for LF */
   if (fork() == 0)
      execl("/bin/csh","csh","-c","stty -echo cbreak",NULL);
   wait(&status);
   if (status != 0)
      exit(1);

   /* initialize arrays & pointers */
   currpage = NULL;
   for (i=0; i<NKEYS; i++)
      keys[i] = SPACECHAR;
   for (i=0; i<XGRIDMAX; i++)
      for (j=0; j<YGRIDMAX; j++)
         pagemap[j][i] = NULL;

   /* create page nodes & add to pagemap */
   for (i=0; i<NPAGES; i++){
      optr = (struct onodetype *)calloc(1,ONODESIZE);
      optr->ord = ORDPAGE0+i;
      optr->oclass = PAGE;
```

```
        optr->status = TABLEAU;
        optr->drawflag = 0;
        strcpy(optr->name,"../Gfiles/page.0");
        optr->name[strlen(optr->name)-1] += i;
        optr->x = XPAGE0; optr->y = YPAGE0+i*PNODESEPN;
        optr->wr = PNODEWD; optr->h = 2*PTABRAD;
        for (j=0; j<TABMAX; j++){
            optr->tab[j].tclass = NOTUSED;
            optr->tab[j].clink = NULL;
            optr->tab[j].dlink = NULL;
            optr->tab[j].ctab = optr->tab[j].dtab = 0;
        } /* for j */
        optr->tab[0].tclass = DATATAB;
        pptr = (struct pnodetype *)calloc(1,PNODESIZE);
        pptr->obj = optr;
        pptr->next = NULL;
        pagemap[i+1][XGRIDMAX-1] = pptr;
    } /* for i */
}
```

```
        /*- - - - - - - - - - - - - - - - - - - - - - - */
        /*  I N I T I A T E   P R O C E S S E S      */
        /*- - - - - - - - - - - - - - - - - - - - - - - */
/*
            Action: Create a process to read from the keyboard in case the
                    next group of processes requires it. Then, initiate
                    execution of the group. If none in the group requires
                    keyboard input, then kill the keyboard reading process.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

initiate (optr,ptrproclist,args)

struct onodetype *optr;
proclisttype *ptrproclist;
struct arglisttype *args;

{
   int rdkbfds[2],rdkbpid,noreaders,status,pid,stat;

   /* DEBUG */
   if (debug) printf("** initiate **");

   /* open a pipe to a read-from-keyboard process */
   /* this pipe should have readfd=5 and writefd=6 */
   if (pipe(rdkbfds)<0){
      printmess("CANNOT CREATE READ KB PIPE");
      return(1);
   }

   /* fork a read-from-keyboard process */
   rdkbpid = fork();
   if (rdkbpid<0){
      printmess("CANNOT FORK READ KB PROCESS");
      return(1);
   }
   if (rdkbpid==0)  /* child */
      rdkbproc();

   /* parent */

   /* DEBUG */
   if (debug) {printf("<r:%d>",rdkbpid); fflush(stdout);}

   /* so far, no processes are reading from keyboard */
   noreaders = TRUE;

   /* execute the next group of concurrent processes */
```

```
    status = executegroup(optr,ptrproclist,args,&noreaders);

    /* close access to rdkb pipe and kill rdkbproc, if no one is using it */
    close(5); close(6);
    if (noreaders){
        kill(rdkbpid,9);
    }
    return(status);
}
```

```
        /*- - - - - - - - -  - - - */
        /*  I N S E R T   L I N K  */
        /*- - - - - - - - - - - - - */
/*
            Action: Controls the insertion of a link between the currently
                    selected tab and one on another object, to be selected
                    by the user. One end of a 'cursor line' is attached to
                    the current node, the other is moved by the user by
                    means of the cursor keys. Once the user selects the
                    second object, this procedure calls 'putlink' in order
                    to choose a particular tab on that object.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

insertlink(tabnum,plinknum,optr)

int tabnum,*plinknum;
struct onodetype *optr;

{
    int loc[2],tab[2],key,done;

    /* DEBUG */
    if (debug) printf("** insertlink **");

    if (optr->tab[tabnum].clink!=NULL && optr->tab[tabnum].dlink!=NULL){
        printmess("NO ROOM ON TAB");
        return(1);
    }
    key = readkey();
    if (key == END)
        return(0);
    gettabloc(tabnum,optr,tab);
    loc[X] = tab[X]; loc[Y] = tab[Y];
    cursorline(tab,loc,DRAW);
    done = FALSE;
    do {
        while (key!=SELECT && key!=END){
            if (iscursor(key))
                movecursline(tab,loc,key);
            else
                printmess("INVALID KEY");
            key = readkey();
        } /* while */
        if (key == SELECT){
            done = putlink(loc,tabnum,plinknum,optr);
            if (!done)
                key = readkey();
```

```
        }
        else if (key == END){
            cursorline(tab,loc,ERASE);
            done = TRUE;
        }
    } while (!done);
    return(0);
}
```

```
        /*-   - - - - - - - - - - - - - - */
        /*  I N S E R T   O B J E C T    */
        /*-   - - - - - - - - - - - - - - */
/*
        Action: Accepts object class (or type) and name from user, calls
                'getnode' to allocate an object-node for this object,
                draws object on the screen, and inserts pointers to it
                into the page map.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern struct onodetype *currpage;

insertobj(loc)

int loc[2];

{
    int key,class;
    char name[NAMEMAX];
    struct onodetype *optr,*getnode();

    /* DEBUG */
    if (debug) printf("** insertobj **");

    if (currpage->status==TEXT){
        printmess("DELETE TEXT BEFORE USING INSERT");
        return(1);
    }

    key = readkey();
    while (isnotobject(key)){
        if (key == END){
            return(0);
        }
        printmess("CHOOSE OBJECT TYPE");
        key = readkey();
    }
    class = key;

    /* read name from keyboard */
    name[0] = ' ';
    if (isnameable(class)){
        cursor(loc,ARROW,DRAW);
        getname(name);
        cursor(loc,ARROW,ERASE);
    }
    else
        strcpy(name,"_n_o_n_a_m_e_");
```

```
    /* create node for this object, fill it with object data */
    optr = NULL;
    if (strlen(name)>0)
        optr = getnode(class,name,loc);
    if (optr == NULL){
        return(1);
    }
    drawobj(optr,DRAW);
    addtomap(optr);
    return(0);
}
```

```
        /*- - - - - - - -  - - - */
        /*  M A R K   N O D E S  */
        /*-  - - - - - - - - -  - */
/*
            Action: Mark as UNBORN all nodes which are potential successors
                    to this node during execution. This is called prior to
                    the execution of a tableau in order to initialize it so
                    that the Gshell can keep track of the execution process.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

marknodes(optr)

struct onodetype *optr;

{
    int i;
    struct onodetype *controllink,*datalink;

    /* DEBUG */
    if (debug) printf("** marknodes **");

    /* mark this node */
    if (optr->oclass!=PAGE)
        optr->status = UNBORN;

    /* clear filenumber field of each tab */
    for (i=0; i<TABMAX; i++){
        optr->tab[i].filenum=NOFILE;
        if (isouttab(optr->tab[i].tclass)){
            controllink = optr->tab[i].clink;
            datalink = optr->tab[i].dlink;
            if (controllink!=NULL)
                marknodes(controllink);
            if (datalink!=NULL && controllink!=datalink)
                marknodes(datalink);
        }
    }
    return(0);
}
```

```
        /*- - - - - - - - - - - -   - - - - - - */
        /*  M O V E   C U R S O R   L I N E    */
        /*- - - - - - - - - - - - - - - - - - - */
/*
           Action: Updates position of cursor line in response to the
                   pressing of a cursor key.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

movecursline(tab,loc,key)

int tab[2],loc[2],key;

{
   cursorline(tab,loc,ERASE);
   switch (key){
   case UPCURSOR:
      if (loc[Y]>=YINF+DELTA) loc[Y] -= DELTA; break;
   case DOWNCURSOR:
      if (loc[Y]<=YSUP-DELTA) loc[Y] += DELTA; break;
   case RIGHTCURSOR:
      if (loc[X]<=XGIGIHI-DELTA) loc[X] += DELTA; break;
   case LEFTCURSOR:
      if (loc[X]>=XINF+DELTA) loc[X] -= DELTA; break;
   }
   cursorline(tab,loc,DRAW);
   return(0);
}
```

```
        /*- - - - - - - - - - - - - - */
        /*  M O V E   C U R S O R   */
        /*- - -   - - - - - -   - - - */
/*
            Action: Updates cursor in response to the pressing of a cursor
                    key.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

movecursor(loc,key,cursorstyle)

int loc[2],key,cursorstyle;

{
    /* DEBUG */
    if (debug) printf("** movecursor **");

    cursor(loc,cursorstyle,ERASE);
    switch (key){
    case UPCURSOR:
        if (loc[Y]>=YINF+DELTA) loc[Y] -= DELTA; break;
    case DOWNCURSOR:
        if (loc[Y]<=YSUP-DELTA) loc[Y] += DELTA; break;
    case RIGHTCURSOR:
        if (loc[X]<=XGIGIHI-DELTA) loc[X] += DELTA; break;
    case LEFTCURSOR:
        if (loc[X]>=XINF+DELTA) loc[X] -= DELTA; break;
    }
    cursor(loc,cursorstyle,DRAW);
    return(0);
}
```

```
        /*- - - - - - - - - -  -   */
        /*  M O V E   O B J E C T   */
        /*- - - - -  -  - - - - - */
/*
            Action: (Invoked by the MOVE key.) Move given object to a
                    position selected by means of the cursor keys. The
                    the movement is indicated by a special 'move' cursor.
                    When the END key is pressed, erase object from its
                    current position and redraw it at the final location
                    of the move cursor.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern struct onodetype *currpage;

moveobj(loc,optr)

int loc[2];
struct onodetype *optr;

{
   int obj[2],key;
   char newname[NAMEMAX],changefname[CMDLEN];

   /* DEBUG */
   if (debug) printf("** moveobj **");

   if (optr->oclass == PAGE){
      printmess("CANNOT MOVE PAGE");
      return(1);
   }
   /* move special "move" cursor */
   obj[X] = loc[X] = optr->x; obj[Y] = loc[Y] = optr->y;
   cursor(loc,CROSS,DRAW);                  /* special cursor for move */
   do {
      key = readkey();
      if (iscursor(key))
         movecursor(loc,key,CROSS);
   } while (key!=END);

   /* move object if necessary */
   cursor(loc,CROSS,ERASE);
   if (obj[X]!=loc[X] || obj[Y]!=loc[Y]){
      drawobj(optr,ERASE);
      removefromap(optr);
      /* store new object location */
      optr->x = loc[X]; optr->y = loc[Y];
      /* change file name, if object is a DPANEL */
      if (optr->oclass == DPANEL){
```

```
            sprintf(newname,"%s.%d.%d",currpage->name,optr->x,optr->y);
            if (fexists(optr->name)){
                sprintf(changefname,"mv %s %s",optr->name,newname);
                system(changefname);
            }
            strcpy(optr->name,newname);
        }
        drawobj(optr,DRAW);
        addtomap(optr);
    }
    return(0);
}
```

```
        /*- - - - - - - - - - - */
        /*  N E X T   L I N K  */
        /*- - - - -   - -   - - */
/*
            Action: Return next tab and or link on current object. Used
                    when selecting tabs or links for insertion or
                    deletion. Cycles through tabs from top of node to
                    bottom and from left to right. At each tab, control
                    is chosen before data link, if both exist and are
                    different.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

nextlink(ptabnum,plinknum,optr)

int *ptabnum,*plinknum;
struct onodetype *optr;

{
   struct onodetype *cptr,*dptr;

   /* DEBUG */
   if (debug) printf("** nextlink **");

   cptr = optr->tab[*ptabnum].clink;
   dptr = optr->tab[*ptabnum].dlink;
   if (*plinknum==0 && dptr!=NULL && dptr!=cptr)
      *plinknum = 1;
   else {
      if (++(*ptabnum)>=TABMAX || optr->tab[*ptabnum].tclass==NOTUSED)
         *ptabnum = 0;
      cptr = optr->tab[*ptabnum].clink;
      dptr = optr->tab[*ptabnum].dlink;
      if (cptr==NULL && dptr!=NULL) *plinknum = 1;
      else *plinknum = 0;
   }
```

```
        /*- - - - - - */
        /*  P A G E   */
        /*- -   - - - */
/*
            Action: Copies tableaux and or other data on current page to
                    page file, clears page map, and erases screen. Copies
                    selected page from its page file, builds page map for
                    selected page, draws new page, and makes it the
                    current page.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern struct onodetype *currpage;

page(loc,optr)

int loc[2];
struct onodetype *optr;

{
   int low[2],high[2];

   /* DEBUG */
   if (debug) printf("** page **");

   if (optr->oclass != PAGE){
      printmess("OBJECT NOT A PAGE TAB");
      return(NOTOK);
   }
   if (optr == currpage){
      printmess("THIS IS CURRENT PAGE");
      return(NOTOK);
   }
   low[X] = low[Y] = 0;
   high[X] = XPAGELIM; high[Y] = YPAGELIM;
   drawmap(low,high,ERASE);
   copyptof(currpage->name);
   clearmap();
   currpage = optr;
   drawframe();
   if (fnotempty(currpage->name))
      copypfromf(currpage->name);
   drawmap(low,high,DRAW);
   return(OK);
}
```

```
        /*- -   - - - - -     - - - - - */
        /*  P R E P A R E   I N T A B   */
        /*- - - - - - - - - - - - - - - */
/*
            Action: Open file or pipe attached to this tab, in preparation
                    for execution of the associated object node.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

prepintab(optr,i,pnoreaders)

struct onodetype *optr;
int i,*pnoreaders;

{
   int pipedescr[2],datatab,fd;
   struct onodetype *datalink,*controllink;

   /* DEBUG */
   if (debug) printf("** prepintab **");

   controllink = optr->tab[i].clink;
   datalink = optr->tab[i].dlink;
   datatab = optr->tab[i].dtab;
   if (datalink!=NULL){
      switch(datalink->oclass){
      case PARAM:
         if (optr->tab[i].filenum<0 &&
         (optr->tab[i].filenum=open(datalink->name,READ))<0){
            printmess("CANNOT OPEN FILE");
            return(NOTOK);
         }
         break;
      case DPANEL:
         if (optr->tab[i].filenum<0 &&
         (optr->tab[i].filenum=open(datalink->name,READ))<0){
            printmess("CANNOT READ FROM EMPTY DATA PANEL");
            return(NOTOK);
         }
         break;
      case PAGE:
         if (optr->tab[i].filenum<0){
            if (fnotempty(datalink->name))
               optr->tab[i].filenum = open(datalink->name,READ);
            else {
               printmess("CANNOT READ FROM EMPTY PAGE");
               return(NOTOK);
            }
```

```
        }
        break;
    case COMMAND:
    case TEE:
        /* if not a pipe, then error */
        if (controllink!=datalink){
            printmess("DATA LINK SHOULD BE PIPE");
            return(NOTOK);
        }
        if (optr->tab[i].filenum<0){
            if (pipe(pipedescr)<0){
                printmess("CANNOT OPEN PIPE");
                return(NOTOK);
            }
            else {
                optr->tab[i].filenum = pipedescr[READ];
                datalink->tab[datatab].filenum = pipedescr[WRITE];
            }
        }
        break;
    default:
        break;
    } /* switch */
} /* if */
else if (optr->oclass!=TEE){  /* and datalink==NULL */
    optr->tab[i].filenum = 5;  /* read end of pipe from rdkbproc */
    *pnoreaders = FALSE;
}
return(OK);
```

```
        /*- - - - - - -  - - - - -  - */
        /*  P R E P A R E   O U T T A B  */
        /*- - - - - - - - - - - - - - - */
/*
            Action: Open file or pipe attached to this tab, in preparation
                    for execution of the associated object node.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

prepouttab(optr,i)

struct onodetype *optr;
int i;

{
   int datatab,pipedescr[2],fd;
   struct onodetype *controllink,*datalink;

   /* DEBUG */
   if (debug) printf("** prepouttab **");

   controllink = optr->tab[i].clink;
   datalink = optr->tab[i].dlink;
   datatab = optr->tab[i].dtab;
   if (datalink!=NULL){
      switch (datalink->oclass){
      case PARAM:
         if (optr->tab[i].filenum<0 &&
         (optr->tab[i].filenum=creat(datalink->name,NORMALMODE))<0){
            printmess("CANNOT CREATE FILE");
            return(NOTOK);
         }
         break;
      case PAGE:
         if (fnotempty(datalink->name)){
            printmess("CANNOT WRITE TO NON-EMPTY PAGE");
            return(NOTOK);
         }
         if (optr->tab[i].filenum<0 &&
         (optr->tab[i].filenum=fd=creat(datalink->name,NORMALMODE))<0){
            printmess("CANNOT CREATE PAGE FILE");
            return(NOTOK);
         }
         break;
      case COMMAND:
      case TEE:
         /* if not a pipe, then error */
         if (controllink!=datalink){
```

```
            printmess("DATA LINK SHOULD BE A PIPE");
            return(NOTOK);
        }
        /* no break here */
    case DPANEL:
        if (optr->tab[i].filenum<0){
            if (pipe(pipedescr)<0){
                printmess("CANNOT OPEN PIPE");
                return(NOTOK);
            }
            else {
                optr->tab[i].filenum = pipedescr[WRITE];
                datalink->tab[datatab].filenum = pipedescr[READ];
            }
        }
        break;
    default:
        break;
    } /* switch */
} /* if */
else if (optr->oclass!=TEE)  /* and datalink==NULL */
    optr->tab[i].filenum = 4;  /* write end of pipe to wrtkbproc */
return(OK);
}
```

```
        /*- - - - - - - - -  - - - - - - - - - - */
        /*  P R I N T   M E S S A G E   A N D   */
        /*      P R I N T   M E S S A G E 2     */
        /*- - - - - - -  -  - -  - -  - - - */
/*
            Action: Print a character string, usually an error message, in
                    the command field at the bottom of the screen.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern int oldmessage;

printmess(message)

char *message;

{
    char *buff,*calloc();

    /* DEBUG */
    if (debug) printf("** printmess **");

    /* allocate buffer space */
    buff = calloc(NAMEMAX,sizeof(char));

    /* clip message */
    *(message+MESSLEN/CHARSEPN) = ' ';

    /* clear message field, if necessary, and print message */
    if (oldmessage){
        sprintf(buff,"P[%d,%d]@RP[,-%d]V[+%d]@QP[%d,%d]@E'%s'",XMESSBEGIN,
        YGIGIHI,BORDERWD2-1,MESSLEN,XMESSBEGIN,YMESSBEGIN,message);
    }
    else {
        sprintf(buff,"@QP[%d,%d]@E'%s'",XMESSBEGIN,YMESSBEGIN,message);
    }
    write(1,buff,strlen(buff));
    cfree(buff);
    oldmessage = TRUE;
    return(0);
}


printmess2(message1,message2)

char *message1,*message2;

{
    strcat(message1,message2);
```

```
    printmess(message1);
    return(0);
}
```

```
        /*- - -  - - - -  -  - -  - - - -*/
        /*  P U T   C O N T R O L   L I N K  */
        /*- - -  - - - - - - - - - -  -  - */
/*
            Action: Insert a control link between the specified tabs on
                    the specified objects. Procedure checks, first, to
                    see whether the insertion is legal.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

putctrllink(tabnum1,plinknum,optr1,tabnum2,optr2)

int tabnum1,*plinknum,tabnum2;
struct onodetype *optr1,*optr2;

{
   int tab1[2],tab2[2];
   char tclass1,tclass2;

   /* DEBUG */
   if (debug) printf("** putctrllink **");

   if (isdataobj(optr1->oclass) || isdataobj(optr2->oclass)){
      printmess("NO CTRL LINK TO DATA OBJECT");
      return(FALSE);
   }
   tclass1 = optr1->tab[tabnum1].tclass;
   tclass2 = optr2->tab[tabnum2].tclass;
   if (areintabs(tclass1,tclass2) || areouttabs(tclass1,tclass2)){
      printmess("LINK IN_TO_OUT OR OUT-TO_IN");
      return(FALSE);
   }
   if (tclass1==FLAGTAB || tclass2==FLAGTAB){
      printmess("NO CTRL LINK TO FLAG TAB");
      return(FALSE);
   }
   if ((optr1->tab[tabnum1].clink!=NULL) || (optr2->tab[tabnum2].clink!=NULL)){
      printmess("DELETE EXISTING LINK");
      return(FALSE);
   }
   optr1->tab[tabnum1].clink = optr2;
   optr1->tab[tabnum1].ctab = tabnum2;
   optr2->tab[tabnum2].clink = optr1;
   optr2->tab[tabnum2].ctab = tabnum1;
   highlight(tabnum1,1,optr1,ERASE);
   highlight(tabnum1,0,optr1,DRAW);
   *plinknum = 0;
   return(TRUE);
}
```

```
        /*-  -  - - - - - - - - -  - - */
        /*  P U T   D A T A   L I N K   */
        /*- - - -  - - - - - - - - - - */
/*
        Action: Insert data link between the specified tabs on the
                specified objects. Procedure checks, first, to see
                whether the insertion is legal.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern struct onodetype *currpage;

putdatalink(tabnum1,plinknum,optr1,tabnum2,optr2)

int tabnum1,*plinknum,tabnum2;
struct onodetype *optr1,*optr2;

{
    int ctab1,ctab2,oclass1,oclass2,tab1[2],tab2[2];
    char tclass1,tclass2;
    struct onodetype *clink1,*clink2;

    /* DEBUG */
    if (debug) printf("** putdatalink **");

    oclass1 = optr1->oclass; oclass2 = optr2->oclass;
    if (isdataobj(oclass1) && isdataobj(oclass2)){
        printmess("CANNOT LINK TWO DATA OBJECTS");
        return(FALSE);
    }
    tclass1 = optr1->tab[tabnum1].tclass;
    tclass2 = optr2->tab[tabnum2].tclass;
    if ((tclass1==FLAGTAB && oclass2!=PARAM) ||
    (tclass2==FLAGTAB && oclass1!=PARAM)){
        printmess("LINK FLAGTAB TO PARAM");
        return(FALSE);
    }
    if ((tclass1==OUTTAB && optr2==currpage) ||
    (tclass2==OUTTAB && optr1==currpage)){
        printmess("CANNOT LINK OUTTAB TO CURRENT PAGE");
        return(FALSE);
    }
    if (issynctab(tclass1) || issynctab(tclass2)){
        printmess("NO DATA LINE TO SYNCTAB");
        return(FALSE);
    }
    if ((optr1->tab[tabnum1].dlink!=NULL) || (optr2->tab[tabnum2].dlink!=NULL)){
        printmess("DELETE EXISTING LINK");
        return(FALSE);
```

```
    }
    if ((!isdataobj(oclass1)) && (!isdataobj(oclass2))){
        if (areintabs(tclass1,tclass2) || areouttabs(tclass1,tclass2)){
            printmess("LINK IN-TO-OUT OR OUT-TO-IN");
            return(FALSE);
        }
        clink1 = optr1->tab[tabnum1].clink; ctab1 = optr1->tab[tabnum1].ctab;
        clink2 = optr2->tab[tabnum2].clink; ctab2 = optr2->tab[tabnum2].ctab;
        if (clink1==NULL && clink2==NULL){
            clink1 = optr1->tab[tabnum1].clink = optr2;
            ctab1 = optr1->tab[tabnum1].ctab = tabnum2;
            clink2 = optr2->tab[tabnum2].clink = optr1;
            ctab2 = optr2->tab[tabnum2].ctab = tabnum1;
        }
        if (clink1!=optr2 || ctab1!=tabnum2 || clink2!=optr1 || ctab2!=tabnum1){
            printmess("DELETE EXISTING LINK");
            return(FALSE);
        }
    }
    /* insert and draw data link */
    optr1->tab[tabnum1].dlink = optr2;
    optr1->tab[tabnum1].dtab = tabnum2;
    optr2->tab[tabnum2].dlink = optr1;
    optr2->tab[tabnum2].dtab = tabnum1;
    highlight(tabnum1,0,optr1,ERASE);
    highlight(tabnum1,1,optr1,DRAW);
    *plinknum = 1;
    return(TRUE);
}
```

```
        /*- - - - - - - - - - */
        /*  P U T   L I N K   */
        /*- - - - - - - - - - */
/*
            Action: Controls the attachment of a data or control link
                    from the specified tab to a tab on an object
                    selected by the user. Allows user to select an
                    object and then select a tab on that object. It
                    then calls 'putdatalink' or 'putctrllink' to
                    perform the actual insertion of the link.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

putlink(loc,tabnum1,plinknum,optr1)

int loc[2],tabnum1,*plinknum;
struct onodetype *optr1;

{
    int nobjects,tabnum2,tab[2],key,done;
    struct onodetype *optr2,*getobj();

    /* DEBUG */
    if (debug) printf("** putlink **");

    done = FALSE;
    optr2 = getobj(loc,&nobjects);
    if (nobjects==0){
        printmess("NOTHING HERE. MOVE CURSOR");
        return(done);
    }
    if (nobjects>1){
        printmess("TOO MANY OBJECTS. MOVE CURSOR");
        return(done);
    }
    tabnum2 = 0;
    gettabloc(tabnum1,optr1,tab);
    cursorline(tab,loc,ERASE);
    gettabloc(tabnum2,optr2,loc);
    do {
        cursorline(tab,loc,DRAW);
        key = readkey();
         while (key==SELECT){
           if (++tabnum2>=TABMAX || optr2->tab[tabnum2].tclass==NOTUSED)
               tabnum2 = 0;
           cursorline(tab,loc,ERASE);
           gettabloc(tabnum2,optr2,loc);
           cursorline(tab,loc,DRAW);
```

```
            key = readkey();
        } /* while */
        cursorline(tab,loc,ERASE);
        if (key==PIPE)
            done = putdatalink(tabnum1,plinknum,optr1,tabnum2,optr2);
        else if (key==CHANNEL)
            done = putctrllink(tabnum1,plinknum,optr1,tabnum2,optr2);
        else if (key==END){
            done = TRUE;
        }
        else
            printmess("INVALID KEY");
    } while (!done);
    return(done);
}
```

```
        /*- - - - - - - - - - - - - - - - -  - -  - -  - - -  - - */
        /*  R E A D - F R O M - K E Y B O A R D   P R O C E S S   */
        /*- - - - - - - - - - - - -  - - - - - -  - -  - - -  - - */
/*
            Action: This process is forked by Gshell, when EXECUTE key
                    is pressed, in order to read keyboard input and pipe
                    it to those object-node in-tabs designated by the
                    user to be receiving their input from the keyboard.
                    The input received by this process is also echoed
                    in the command field at the bottom of the screen.
                    Note: if more than one process attempts to read from
                    the keyboard at the same time, the input is shared
                    randomly between them.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

rdkbproc()

{
    int i,locx;
    char ch,*buff,*calloc();

    /* DEBUG */
    if (debug) printf("** rdkbproc **");

    /* close all file descriptors, other than 0, 1, 2, and 6 */
    /* file descriptor 6 is the write end of the rdkb pipe */
    for (i=3; i<20; i++){
        if (i!=6)
            close(i);
    }

    /* allocate buffer space for output */
    buff = calloc(NAMEMAX,sizeof(char));

    /* read from keyboard and write to rdkb pipe, also, echo input */
    /* to the Gshell screen message field */
    locx = XMESSBEGIN;
    while ((ch=getchar())!=EXITCHAR){
        /* write to pipe */
        write(6,&ch,1);
        if (ch==TABCHAR) ch = SPACECHAR;
        if (ch==NEWLINE){
            /* new line: clear message field and return to left end */
            locx = XMESSBEGIN;
            sprintf(buff,"P[%d,%d]@RP[,-%d]V[+%d]@Q",XMESSBEGIN,YGIGIHI,
            BORDERWD2-1,MESSLEN);
            write(1,buff,strlen(buff));
```

```
        }
        else if (locx<XMESSBEGIN+MESSLEN && ch>=SPACECHAR && ch<=TILDECHAR){
            /* print character if visible */
            sprintf(buff,"P[%d,%d]@E'%c'",locx,YMESSBEGIN,ch);
            write(1,buff,strlen(buff));
            locx += CHARSEPN;
        }
    } /* while */
    close(6);  /* close write end of rdkb pipe */
    cfree(buff);
    exit(0);
}
```

```
        /*- - - - - - - - - - */
        /*  R E A D   K E Y   */
        /*- - - - - - - - -  - */
/*
            Action: Read input from one keystroke. This is not to be
                    confused with 'rdkbproc'. The latter provides
                    input to a Gshell tableau during execution. This
                    process analyses the character strings (up to 3
                    ascii characters) produced by single keys of the
                    GIGI terminal's standard keyboard, the cursor
                    keys, or the programmable keypad, and passes an
                    internal keycode to the Gshell command processor,
                    'doGshellcmds'.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern int oldmessage;
extern int keys[NKEYS];

readkey()

{
   int key,i;
   char *curswitch;

   /* DEBUG */
   if (debug) printf("** readkey **");

   /* ANSI keypad codes: ESC char char */
   if ((key = getchar()&MAXASCII) == ESCCHAR){
      getchar();       /* skip middle character */
      key = getchar()|MAXASCPL1;
      if (!iskeypad(key) && !iscursor(key)) key = SPACECHAR;
   }
   /* switch cursor off if the next key is not a cursor key */
   if (!iscursor(key)){
      curswitch = GIGICURSOROFF; write(1,curswitch,strlen(curswitch));
   }

   /* add most recent keycode to history queue */
   /* if it is a Gshell command key or a control key */
   if (!iscursor(key) && !(SPACECHAR<=key && key<=TILDECHAR) && key!=BACKSPACE) {
      for (i=1; i<NKEYS; i++)
         keys[i-1] = keys[i];
      keys[NKEYSMN1] = key;
      echokeys();
   }
   return(key);
}
```

```
        /*- - - - - - - - - - - - - -   - - - - - - - - - -   - - */
        /*  R E A S S I G N   F I L E   D E S C R I P T O R S   */
        /*- - - - - - - - - - - - - - - - - - - - - - -   -   - - - */
/*
          Action: As soon as the process for this node has been forked
                  and before the UNIX command associated with it is
                  executed, reassign the file descriptors used for I/O
                  via the node's tabs. These have been assigned arbit-
                  rarily by means of calls to 'pipe'. They must be
                  changed through calls to 'dup2' so as to fit the
                  Gshell convention: 0,2,4,..., from left to right, for
                  input and 1,3,5,..., from left to right, for output.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

reassignfds(optr)

struct onodetype *optr;

{
   int i,j,k,jbase,fd,properfd,currfd,newfd;
   char tabclass;
   struct {
      int used;            /* this file descriptor is being used */
      int tab;             /* tab of object, *optr, to which it is assigned */
   } fds[MAXFDS];          /* one record for each available file descriptor */

   /* DEBUG */
   if (debug) {printf("** reassignfds **"); fflush(stdout);}

   /* record the tab numbers associated with file descriptors which */
   /* are required and close those file descriptors which are not required */
   for (i=0; i<20; i++){
      fds[i].used = FALSE; fds[i].tab = -1;
   }
   for (j=0; j<TABMAX; j++)
      if ((fd=optr->tab[j].filenum)>0){
         fds[fd].used = TRUE; fds[fd].tab = j;
   }
   /* reassign standard error */
   dup2(2,19); fds[19].used = TRUE; fds[19].tab = -1;
   /* close all file descriptors not used */
   for (i=0; i<19; i++){
      if (!fds[i].used)
         close(i);
   }

   /* reassign file descriptors so that intabs use fds 0, 2, 4 etc. */
```

```
    /* and outtabs use fds 1, 3, 5, etc. */
    /* NOTE: all commands called by Gshell must observe this convention */
    /* for the assignment of file descriptors */
    jbase = 0;
    for (j=0; j<TABMAX; j++){
        if ((tabclass=optr->tab[j].tclass)==INTAB || tabclass==OUTTAB){
            if (tabclass==INTAB)
                properfd = 2*j;
            else {
                if (jbase==0)
                    jbase = j;
                properfd = 2*(j-jbase)+1;
            }
            currfd = optr->tab[j].filenum;
            /* reassign fd to tab j, if it is not the proper one for this tab */
            if (currfd!=properfd && currfd!=NOFILE){
                /* if properfd is being used, duplicate it */
                if ((k=fds[properfd].tab)>0){
                    if ((newfd=dup(properfd))<0){
                        printmess("CANNOT DUPLICATE FILE DESCRIPTOR");
                        exit(NOTOK);
                    }
                    optr->tab[k].filenum = newfd;
                    fds[newfd].used = TRUE;
                    fds[newfd].tab = k;
                }
                dup2(currfd,properfd);
                optr->tab[j].filenum = properfd;
                fds[properfd].used = TRUE;
                fds[properfd].tab = j;
                close(currfd);
                fds[currfd].used = FALSE;
                fds[currfd].tab = -1;
            } /* if */
        } /* if */
    } /* for */
    return(0);
}
```

```
        /*- - - - - - - - - */
        /*  R E F R E S H  */
        /*- - - - - - - - - */
/*
            Action: Redraw specified object or page. Useful for cleaning
                    up after a number of erasures have fragmented the
                    display.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern struct onodetype *currpage;

refresh(loc,optr)

int loc[2];
struct onodetype *optr;

{
   int low[2],high[2];

   /* DEBUG */
   if (debug) printf("** refresh **");

   if (optr->oclass  == PAGE){
      if (optr==currpage){
         if (currpage->status==TABLEAU){
            low[X] = low[Y] = 0;
            high[X] = XPAGELIM; high[Y] = YPAGELIM;
            drawmap(low,high,ERASE);
            drawframe();
            drawmap(low,high,DRAW);
         }
         else
            printmess("REFRESH NOT IMPLEMENTED FOR TEXT");
      }
      else {
         printmess("CHOOSE CURRENT PAGE");
      }
   }
   else
      drawobj(optr,DRAW);
   return(0);
}
```

```
        /*- - - - - - - - - - - - - - -   - - */
        /*  R E M O V E   F R O M   M A P   */
        /*- - - - - - - - - - - - - - - - - - */
/*
            Action: Remove all pointers to the specified object from the
                    page map.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;
extern struct pnodetype *pagemap[YGRIDMAX][XGRIDMAX];

removefromap(optr)

struct onodetype *optr;

{
   int i,j,low[2],high[2];
   struct pnodetype *oldptr,*pptr;

   /* DEBUG */
   if (debug) printf("** removefromap **");

   getextentobj(optr,low,high);
   for (i=low[X]; i<=high[X]; i++){
      for (j=low[Y]; j<=high[Y]; j++){
         oldptr = NULL; pptr = pagemap[j][i];
         while (pptr!=NULL && pptr->obj!=optr){
            oldptr = pptr;
            pptr = pptr->next;
         } /* while */
         if (pptr->obj == optr) {
            if (oldptr == NULL)
               pagemap[j][i] = pptr->next;
            else
               oldptr->next = pptr->next;
            cfree((char *)pptr);
         } /* if */
      } /* for j */
   } /* for i */
   return(0);
}
```

```
        /*- - - -  -  - - - - - - - - */
        /*  REMOVE  LINKS  */
        /*- -  - - -  - - - - - -  */
/*
        Action: Remove all links from this object to any other object.
                Note: this involves removing pointers from this node
                as well as from those nodes to which it is linked.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

removelinks(optr)

struct onodetype *optr;

{
    int i,tabnum;
    struct onodetype *cptr,*dptr;

    /* DEBUG */
    if (debug) printf("** removelinks **");

    for (i=0; i<TABMAX; i++){
        if ((cptr=optr->tab[i].clink) != NULL){
            tabnum = optr->tab[i].ctab;
            cptr->tab[tabnum].clink = NULL;
            cptr->tab[tabnum].ctab = 0;
        }
        if ((dptr=optr->tab[i].dlink) != NULL){
            tabnum = optr->tab[i].dtab;
            dptr->tab[tabnum].dlink = NULL;
            dptr->tab[tabnum].dtab = 0;
        }
    }
    return(0);
}
```

```
        /*- - - - - - - - - - - */
        /*    S E C T O R S     */
        /*- - - - - - - - - - - */
/*
        Action: Procedures to draw each of the four 90 degree sectors
                of a circle (unshaded) or disk (shaded).
*/

#include "Gshell.defs"

        /* globals */
extern int debug;

sectNW(center,radius,shade,blink,mode)

int center[2],radius;
char shade,blink,mode;

{
    char c;

    /* DEBUG */
    if (debug) printf("** sectNW **");

    if (visible(center[X]-radius,center[Y]-radius,radius)){
        sectsetup(center,radius,shade,blink,mode);
        c = '2'; write(1,&c,1);
    }
}


sectSW(center,radius,shade,blink,mode)

int center[2],radius;
char shade,blink,mode;

{
    char c;

    /* DEBUG */
    if (debug) printf("** sectSW **");

    if (visible(center[X]-radius,center[Y],radius)){
        sectsetup(center,radius,shade,blink,mode);
        c = '4'; write(1,&c,1);
    }
}


sectSE(center,radius,shade,blink,mode)

int center[2],radius;
```

```
char shade,blink,mode;

{
    char c;

    /* DEBUG */
    if (debug) printf("** sectSE **");

    if (visible(center[X],center[Y],radius)){
        sectsetup(center,radius,shade,blink,mode);
        c = '6'; write(1,&c,1);
    }
}


sectNE(center,radius,shade,blink,mode)

int center[2],radius;
char shade,blink,mode;

{
    char c;

    /* DEBUG */
    if (debug) printf("** sectNE **");

    if (visible(center[X],center[Y]-radius,radius)){
        sectsetup(center,radius,shade,blink,mode);
        c = '0'; write(1,&c,1);
    }
}


sectsetup(center,radius,shade,blink,mode)

int center[2],radius;
char shade,blink,mode;

{
    char *buff,*calloc();
    int selector;

    buff = calloc(16,sizeof(buff));
    sprintf(buff,"P[%d,%d]",center[X],center[Y]); write(1,buff,strlen(buff));
    /* The following is intended to speed up GIGI processing */
    selector = (shade==SHADEON)*4 + (blink==BLINKON)*2 + (mode==DRAW);
    switch (selector){
    case 0: sprintf(buff,"@S"); write(1,buff,strlen(buff)); break;
    case 1: sprintf(buff,"@T"); write(1,buff,strlen(buff)); break;
    case 2: sprintf(buff,"@U"); write(1,buff,strlen(buff)); break;
    case 3: sprintf(buff,"@V"); write(1,buff,strlen(buff)); break;
    case 4: sprintf(buff,"@W"); write(1,buff,strlen(buff)); break;
    case 5: sprintf(buff,"@X"); write(1,buff,strlen(buff)); break;
```

```
    case 6: sprintf(buff,"@Y"); write(1,buff,strlen(buff)); break;
    case 7: sprintf(buff,"@Z"); write(1,buff,strlen(buff)); break;
    }
    sprintf(buff,"%d),A90)",radius); write(1,buff,strlen(buff));
    cfree(buff);
}
```

```
        /*- - - - - - - - - */
        /*  T I D Y   U P   */
        /*-   - - - -   - - */
/*
            Action: Perform housekeeping after forking the process asso-
                    ciated with a command node, during execution of a
                    tableau.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

tidyup(optr,procid,ptrproclist)

struct onodetype *optr;
int procid;
proclisttype *ptrproclist;

{
   int i;
   char *calloc();
   struct procnodetype *procptr;

   /* DEBUG */
   if (debug) printf("** tidyup **");

   /* add process to process list */
   procptr = (struct procnodetype *)calloc(1,PRCNODESIZE);
   procptr->process = procid;
   procptr->object = optr;
   procptr->nextproc = *ptrproclist;
   *ptrproclist = procptr;

   /* close parent's access to file descriptors opened for */
   /* the process just added to the process list */
   /* Note: file descriptors 0-6 must be kept open */
   if (optr!=NULL){
      for (i=0; i<TABMAX; i++){
         if (optr->tab[i].filenum>6)
            close(optr->tab[i].filenum);
      } /* for */
   } /* if */
}
```

```
        /*- -   - - - - - */
        /*  WI NDOW   */
        /*- - - - - - - - */
/*
        Action: Change size of data panel, as specified by user by means
                of cursor keys.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

window(loc,optr)

int loc[2];
struct onodetype *optr;

{
    int diag[2],width,height,newwidth,newheight;
    int key;

    /* DEBUG */
    if (debug) printf("** window **");

    if (optr->oclass != DPANEL){
        printmess("WINDOW: DATA PANELS ONLY");
        return(1);
    }
    /* move special "window" cursor */
    loc[X] = optr->x; loc[Y] = optr->y;
    width = optr->wr; height = optr->h;
    diag[X] = loc[X]+width; diag[Y] = loc[Y]+height;
    cursor(diag,ANGLE,DRAW);              /* special "window" cursor */
    do {
        key = readkey();
        if (iscursor(key)  && diag[X]>loc[X] && diag[Y]>loc[Y])
            movecursor(diag,key,ANGLE);
    } while (key != END);

    /* change data panel size if necessary */
    newwidth = diag[X]-loc[X]; newheight = diag[Y]-loc[Y];
    cursor(diag,ANGLE,ERASE);
    if (newwidth!=width || newheight!=height){
        drawobj(optr,ERASE);
        removefromap(optr);
        optr->wr = newwidth; optr->h = newheight;
        drawobj(optr,DRAW);
        addtomap(optr);
    }
    return(0);
}
```

```
        /*- - - - - - - -  - - - - - - - - - - - - - - - - - - - - */
        /*  W R I T E - T O - K E Y B O A R D   P R O C E S S    */
        /*- - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
            Action: This process is forked by Gshell, when EXECUTE key is
                    pressed, in order to handle output directed to the
                    keyboard by processes of the tableau being executed.
                    The output of this process is displayed in the command
                    field at the bottom of the screen. Note: this output
                    should not be confused with output to a data panel
                    even though the latter is also displayed on the
                    terminal screen.

*/

#include "Gshell.defs"

        /* globals */
extern int debug;

wrtkbproc()

{
   int i,locx,rstatus;
   char ch,*buff,*calloc();

   /* DEBUG */
   if (debug) printf("** wrtkbproc **");

   /* close all file descriptors, other than 0, 1, 2, and 3 */
   /* file descriptor 3 is the read end of the wrtkb pipe */
   for (i=4; i<20; i++){
      close(i);
   }

   /* allocate buffer space for output */
   buff = calloc(NAMEMAX,sizeof(char));

   /* read from pipe and print in the message field of the Gshell screen */
   locx = XMESSBEGIN;
   while ((rstatus=read(3,&ch,1))>0){
      if (ch==TABCHAR) ch = SPACECHAR;
      if (ch==NEWLINE)
         locx = XMESSBEGIN;
      else if (ch>=SPACECHAR && ch<=TILDECHAR){
         if (locx==XMESSBEGIN){   /* clear message field */
            sprintf(buff,"P[%d,%d]@RP[,-%d]V[+%d]@Q",XMESSBEGIN,YGIGIHI,
            BORDERWD2-1,MESSLEN);
            write(1,buff,strlen(buff));
         }
         if (locx<XMESSBEGIN+MESSLEN){
            sprintf(buff,"P[%d,%d]@E'%c'",locx,YMESSBEGIN,ch);
            write(1,buff,strlen(buff));
```

```
                locx += CHARSEPN;
            }
        } /* else */
    } /* while */
    close(3);  /* close read end of wrtkb pipe */
    cfree(buff);
    exit(-rstatus);
}
```