Motorola Mobility, Inc. v. Microsoft Corporation

Doc. 123 Att. 14

EXHIBIT 35

Dockets.Justia.com

US005502839A

United States Patent [19]
Kolnick

[11] **Patent Number: 5,502,839**
[45] **Date of Patent: Mar. 26, 1996**

[54] **OBJECT-ORIENTED SOFTWARE ARCHITECTURE SUPPORTING INPUT/OUTPUT DEVICE INDEPENDENCE**

[75] Inventor: **Frank C. Kolnick**, Willowdale, Canada

[73] Assignee: **Motorola, Inc.**, Schaumburg, Ill.

[21] Appl. No.: **361,738**

[22] Filed: **Jun. 2, 1989**

Related U.S. Application Data

[63] Continuation of Ser. No. 619, Jan. 5, 1987, abandoned.

[51] **Int. Cl.**[6] **G06F 13/00**
[52] **U.S. Cl.** **395/800**; 364/228.2; 364/237.9; 364/239.9; 364/280; 364/284.2; 364/DIG. 1
[58] **Field of Search** .... 364/200 MS File, 364/900 MS File; 395/500

[56] **References Cited**

U.S. PATENT DOCUMENTS

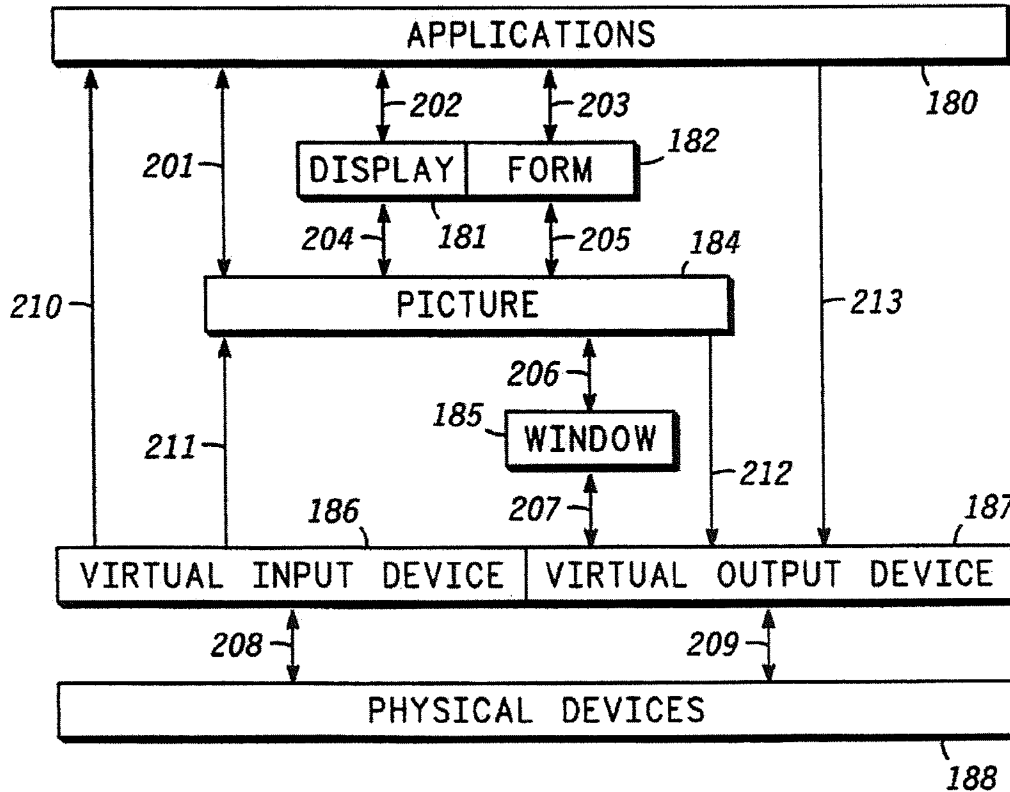
3,930,232	12/1975	Wallach et al.	395/500
4,241,341	12/1980	Thorson	340/747
4,454,593	6/1984	Fleming	364/900
4,485,439	11/1984	Rothstein	395/500
4,547,628	10/1985	Tamura et al.	340/747
4,555,775	11/1985	Pike	364/900
4,559,614	12/1985	Peek et al.	364/900
4,642,790	2/1987	Minshull et al.	364/900
4,754,395	6/1988	Weisshaar et al.	364/200
4,800,523	1/1989	Gerety et al.	395/500
4,858,114	8/1989	Heath et al.	395/500
5,063,494	11/1991	Davidowski et al.	395/800

Primary Examiner—Kevin J. Teska
Assistant Examiner—Ayni Mohamed
Attorney, Agent, or Firm—Walter W. Nielsen; Harold C. McGurk; S. Kevin Pickens

[57] **ABSTRACT**

An object-oriented software architecture interacts with "real" input/output devices exclusively through "virtual" input/output devices. Since all human interface with the operating system is performed through such virtual devices, the system can accept any form of real input or output devices. The lowest level of the operating system converts input from any physical device to virtual form and converts virtual output into suitable physical output. Any number of physical devices can be connected to, removed from, or replaced in the system without disrupting the system.

23 Claims, 9 Drawing Sheets



CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

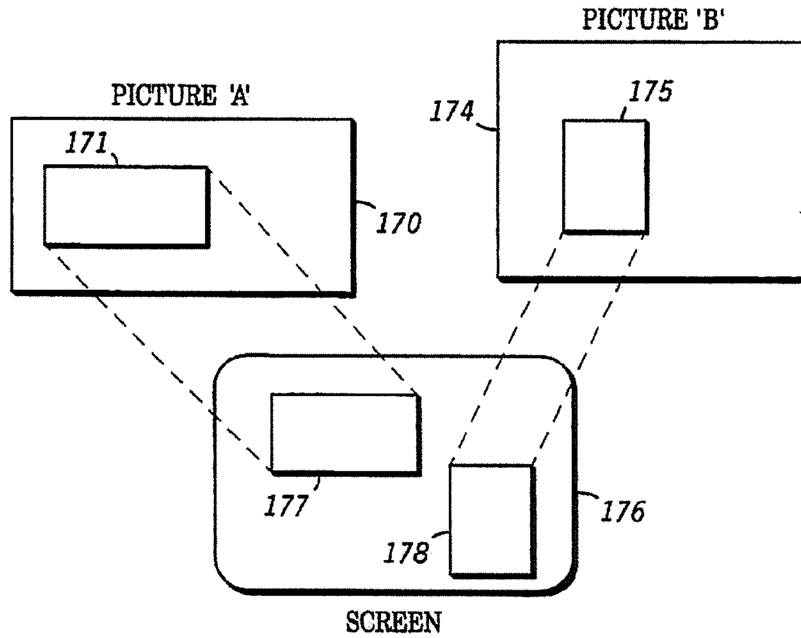
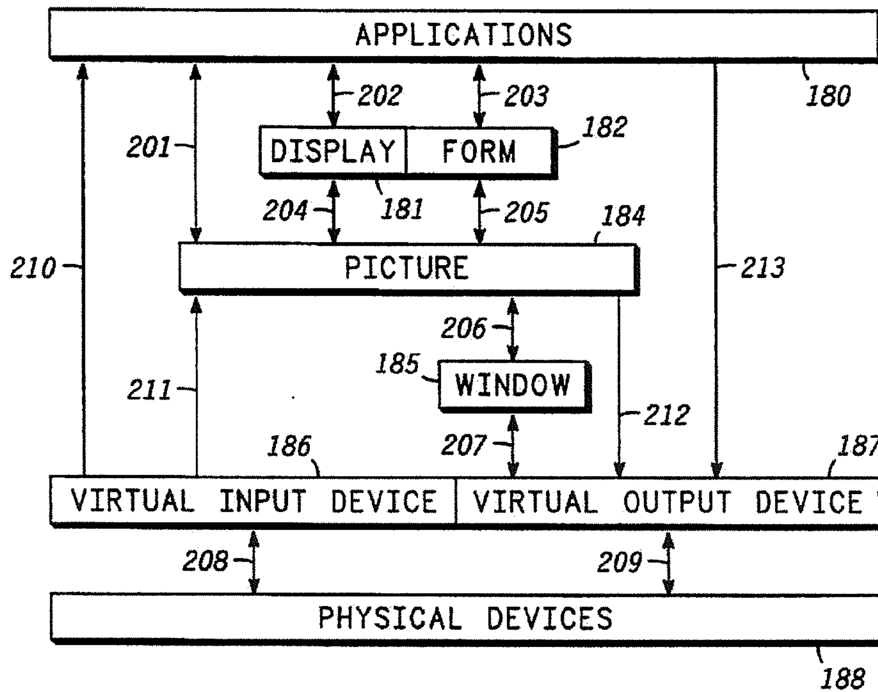


FIG. 7

FIG. 8



CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS
INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019500

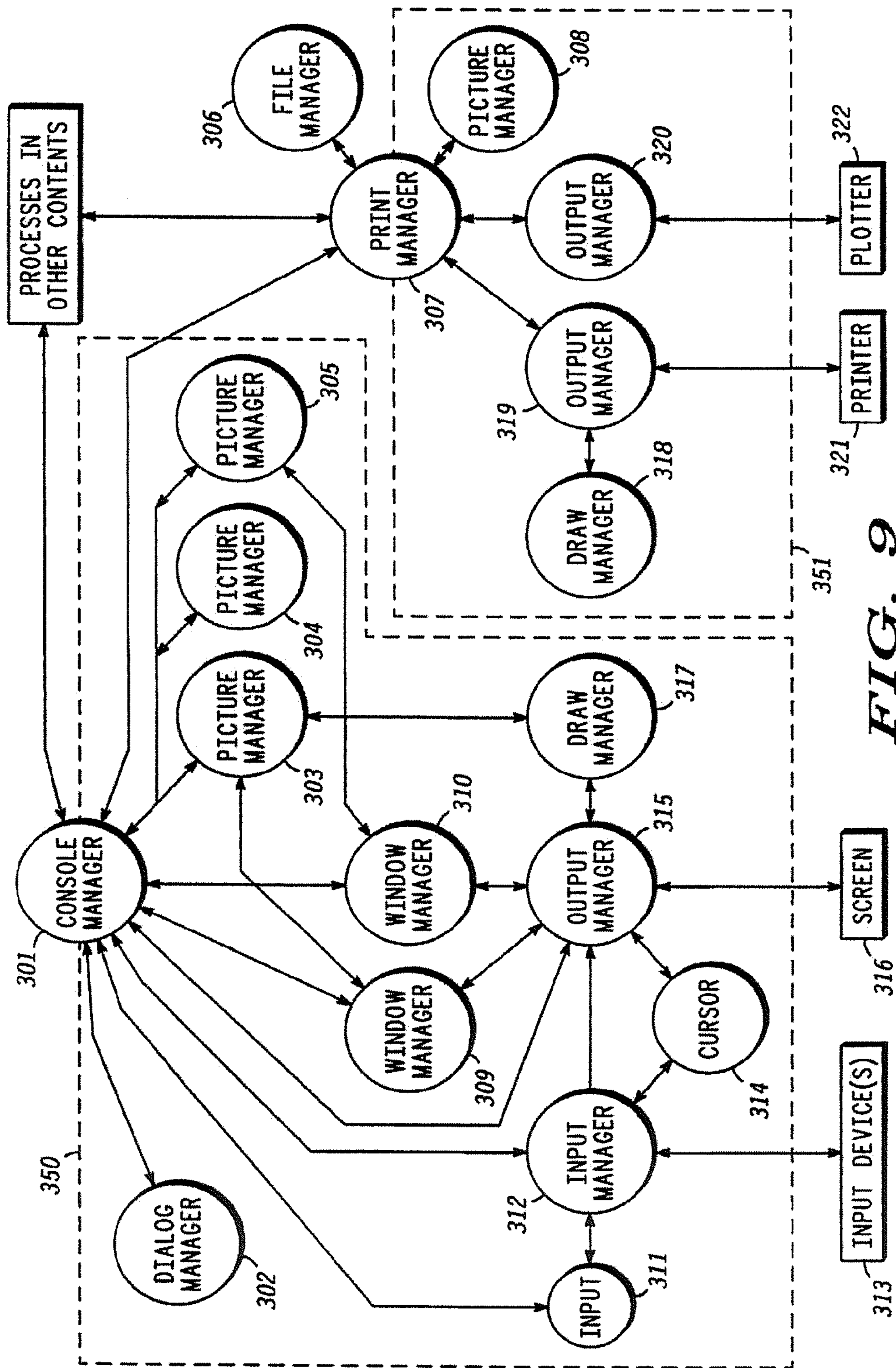


FIG. 9

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019501

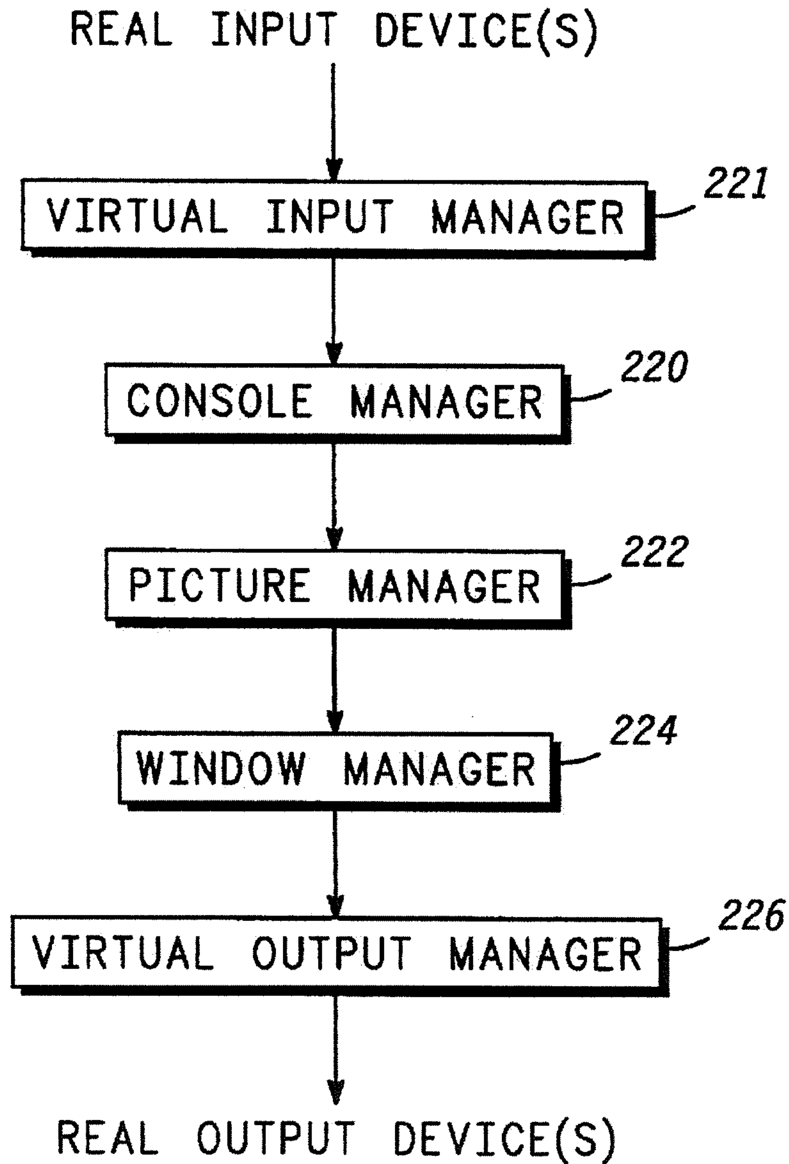


FIG. 12

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS
INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019503

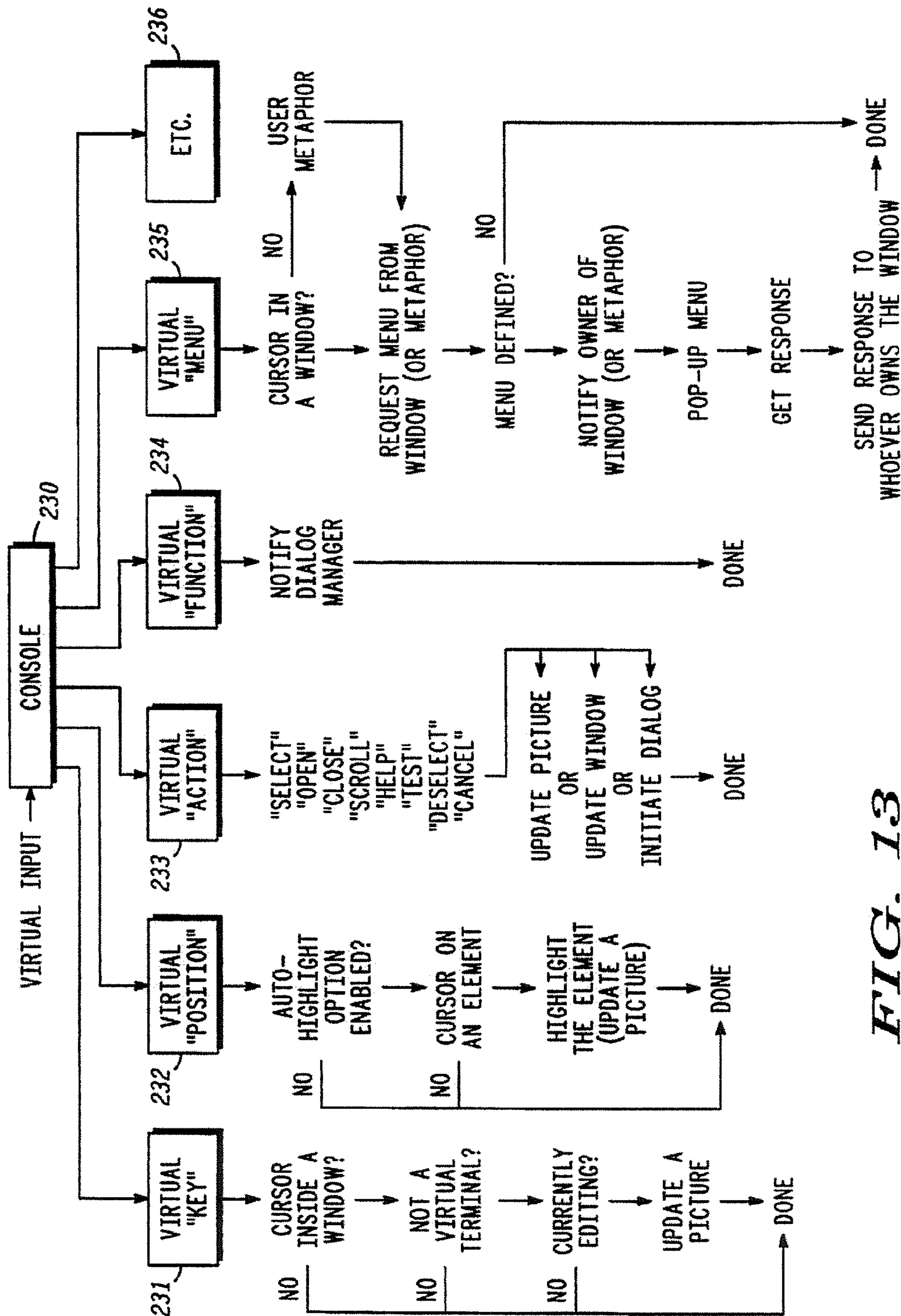


FIG. 13

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS
INFORMATION, SUBJECT TO PROTECTIVE ORDER

"sentences" which describe events and which the Human Interface can use to parse in order to manipulate events for specific requests. For example, events can be dynamically displayed on a screen by time and/or priority, or they can be scanned for a particular "subject" or "object" or any other attribute. Each event can be time-stamped by the sender; if not, it is automatically time-stamped upon receipt.

The Human Interface records all of its own actions, such as printing a report or signing-on a user, and it provides this service to any applications program. In addition, the Human Interface can be requested to trigger any given action upon the occurrence of any given event, thus providing a kind of closed-loop control service to applications.

Modularity—The Human Interface comprises a number of separate software components which can be replicated and distributed throughout the hardware configuration to achieve optimal performance. For example, each time a new "console" (for example, keyboard plus screen) is connected to the system, a new "Console" component is created to manage it. There is no logical limit to the number of consoles that the Human Interface can handle. In general the relevant software component is located close to the hardware or other resources on which it most depends.

HUMAN INTERFACE—BASIC COMPONENTS

The Human interface comprises the following basic components:

Console Manager—It is the central component of a Console context and consequently is the only manager which knows all about its particular "console" It is therefore aware of all screens and keyboards, all windows, and all pictures. Its primary responsibility is to coordinate the activities of the context. This consists of starting up the console (initializing the device managers, etc.) creating and destroying pictures, and allocating and controlling windows for processes in the Human Interface and elsewhere. Thus all access to a console must be indirect, through the relevant Console Manager.

The Console Manager also implements the first level of Human Interface interaction, via menus, prompts, etc., so that applications processes don't have to. Rather than using built-in text and icons, it depends upon the Dialog Manager to provide it with the visible features of the system. Thus all cultural and user idiosyncracies (such as language) are hidden from the rest of the Human Interface.

A Console Manager knows about the following processes: the Output Manager(s) in its context, the Input Manager in its context, the Window Managers in its context, the Picture Managers in its context, and the Dialog Manager in its context. The following processes know about the Console Manager: any one that wants to.

When a Console Manager is started, it waits for the basic processes needed to communicate with the user to start up and "sign on". It this is successful, it is ready to talk to users and other processes (i.e., accept messages from the Input Manager and other processes). All other permanent processes in the context (Dialog, etc.) are assumed to be activated by the system start-up procedure. The "In" and "Cursor" processes (see "Input Manager" and "Output Manager" below) are created by the Console Manager at this time.

The Console Manager generally clears the entire screen and displays appropriate status text during the course of the start-up (by sending picture elements directly to its Output Manager(s)). If any part of the start-up fails, it displays appropriate "error" text and possibly waits for corrective action from a user.

The Console Manager views the screen as being composed of blank (unused) space, windows, and icons. Whenever an input character is received, the Console Manager determines how to handle it depending upon the location of the cursor and the type of input, as follows:

A. Requests to create or eliminate a window are handled within the Console Manager. A window may be opened anywhere on the screen, even on top of another window. A new Picture Manager and possibly a Window Manager may be created as a result, and one or more new messages may be generated and sent to them, or the manager(s) may be told to quit.

B. Icons can only be selected, then moved or opened. The Console Manager handles selection and movement directly. It sends notification of an "open" to the Dialog Manager, which sends a notification to the application process associated with the icon and possibly opens a default window for it.

C. For window-dependent actions, if the cursor is outside all windows, the input is illegal, and the Console Manager informs the user; otherwise the input is accepted. Request which affect the window itself (such as "scroll" or "zoom") are handled directly by the Console Manager. A "select" request is pre-checked, the relevant picture elements are selected (by sending a message to the relevant Picture Manager), and the message is passed on to the process currently responsible for the window. All other inputs are passed directly to the responsible process without being pre-checked.

If the cursor is on a window's frame, the only valid actions are to move, close, or change the dimensions of the window, or select an object in the frame (such as a menu or a scroll bar). These are handled directly by the Console Manager.

D. Requests for Human interface services not in the Console context are treated as errors.

A new window is opened by creating a new Window Manager process and telling it its dimensions and the location of its upper left corner on the screen. It must also be given the PID of a Picture Manager and the coordinates of the part of the picture it is to display, along with the dimensions of a "clipping polygon", if that information is available. (It is not possible to create a window without a picture.) The type and contents of the window frame are also specified. Any of these parameters may be changed at any time.

A new instance of a picture is created by creating a new Picture Manager process with the appropriate name and, optionally, telling it the name of a "file" from which to get its picture elements. If a file is not provided, an "empty" picture is created, with the expectation that picture-drawing requests will fill it in.

Menus, prompts, help messages, error text, and icons are simply predefined pictures (provided through the Dialog Manager) which the Console Manager uses to interact with users. They can therefore be created and edited to meet the requirements of any particular system the same way any picture can be created and edited. Menus and help text are usually displayed on request, although they may sometimes be a result of another operation.

Prompts are displayed when the system needs information from the user. Error text is displayed whenever the user tries to do something that is illegal or when the system is having

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

problems of its own (e.g. "printer out of paper"). Icons are displayed by the Console Manager automatically when a specific frame of reference is requested by the user. The Console Manager may also display informational messages (such as "console starting up") which are automatically erased when the associated action is finished.

Picture Manager—It is created when a picture is built, and it exits when the picture is no longer required. There is one Picture Manager per picture. The Picture Manager constructs a device-independent representation of a picture using a small set of elemental "picture elements" and controls modification and retrieval of the elements.

A Picture Manager knows about the following processes: the process which created it, and the Draw Manager. The following processes know about the Picture Manager: the Console Manager in the same context, and Window Managers in the same context.

A Picture Manager is created to handle exactly one picture, and it need only be created when that picture is being accessed. It can be told to quit at any time, deleting its representation of the picture. Some other process must copy the picture to a file if it needs to be saved.

When a Picture Manager first starts up, its internal picture is empty. It must receive a "load file" request, or a series of "draw" requests, before a picture is actually available. Until that is done any requests which refer to specific elements or locations in the picture will receive an appropriate "not found" status message.

A picture is logically composed of device-independent "elements", such as text, line, arc, and symbol. In general, there is a small number of such elements. Each element consists of a common header, which includes the element's position in the picture's coordinate system, its color, size, etc., and a "value" which is unique to the element's type (e.g. a character string, etc.). The header also specifies how the element combines with other elements in the picture (overlays them, merges with them, etc.). A special element type called "null" is also supported to facilitate the removal of picture elements from pictures or other similar large lists without forcing time-consuming compaction procedures. Any element can therefore be redefined to "null", indicating that it should be ignored for all future processing.

The "null" color (zero) is treated as transparent when used in either the foreground or the background. Specifically, if the foreground color is null, the element itself is not drawn, but it may still be filled in. If the background color is null, the element is not filled in. If the shading pattern is null, and the color is not null, the background fill is solid.

A picture is represented in an internal format which may be different from the external representation of picture elements and which is, in any case, hidden from other processes. This representation is designed to optimize retrieval of picture elements, with a secondary emphasis on adding new elements and modifying or erasing old ones. The order in which the elements were originally drawn is preserved (unless explicit "order" requests have been received to re-arrange them).

Requests to "animate" an element result in the creation of a separate, local "animate" process which performs the necessary transformations and sends the appropriate requests (usually "draw" or "erase") back to the Picture Manager periodically.

A Picture Manager processes incoming requests one at at time, as it receives them. Each message can change the state of the picture for later requests. The Picture Manager supports numerous operations, including the following: "draw" new elements; "modify", "overwrite", or "erase" existing elements; "copy" or "move" elements to another location in the same picture or to any other given process; "group" elements together into one (or "ungroup" them); "scale" them (i.e. expand, stretch, or shrink them); and "rotate" them. It can also be asked to "notify" a particular process if any elements within a given rectangular area (the "viewport") are changed and to determined whether a given location coincides (or come close to) any element in the picture. Any response to a request (e.g., multiple picture elements) is sent in a single message.

When an element is sent as the result of an outstanding "notify" request, all elements which overlap it (and all elements which overlap those elements) are sent as well. These are sent together in one message. The background is displayed by generating a "rectangle" element of the same size as the current viewport with a null foreground color and the appropriate background pattern and color. This element is always the lowest level in the picture; i.e., it is sent before all others. All erasure of elements from a display is accomplished by "draw" requests which redisplay the background and/or elements in the picture, overwriting the "erased" elements. There is no explicit "erase" request to a window (or output) manager.

Input Manager—There is one Input Manager per set of "logical input devices" (such as keyboards, mice, light pens, etc.) connected to the system. The Input Manager handles input interrupts and passes them to the console manager. Cursor movement inputs may also be sent to a designated output manager.

The Input Manager knows about the following processes: the process which initialized it, and possibly one particular Output Manager in the same context. The following process knows about the Input Manager: the Console Manager in the same context.

An Input Manager is created (automatically, at system start-up) for each set of "logical input devices" in the system, thus implementing a single "virtual keyboard" There can only be one such set, and therefore one Input Manager, per Console context. The software (message) interface to each manager is identical, although their internal behavior is dependent upon the physical device(s) to which they communicate. All input devices interrupt service routines (including mouse, digitizing pad, etc.) are contained in Input Managers and hidden from other processes. When ready, each Input Manager must send an "I'm here" message to the closest process named "Console".

An Input Manager must be explicitly initialized and told to proceed before it can begin to process input interrupts. Both of these are performed using appropriate messages. Whichever process initializes the manager becomes tightly coupled to it, i.e., they can exchange messages via PID's rather than by name. The Input Manager will send all inputs to this process (usually the Console Manager). This coupling cannot be changed dynamically; the manager would have to be re-initialized, Between the "initialize" and the "proceed" an Input Manager may be sent one or more "set" requests to define its behavior. It does not need to be able to interpret the meaning of any input beyond distinguishing cursor from non-cursor. Device-independent parameters (such as pixel size and density) are not down-loaded but rather are assumed to be built into the software, some part of which, in general, must be unique to each type of Input Manager.

An Input Manager can be dynamically "linked" to a particular Output Manager, if desired. If so, all cursor control input (or any other given subset of the character set) will be sent to that manager, in addition to the initializing process, as it is received. This assignment can be changed or

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019514

about the Window Manager: the Console Manager in the same context.

The Window Manager's main job is to copy picture elements from a given rectangular area of a picture to a rectangular area (called a "window") on a particular screen. To do so it interacts with exactly one Picture Manager and one Output Manager. A Window Manager need only be created when a window is "opened" on the screen and can be told to quit when the window is "closed" (without affecting the associated picture). When opened, the Window Manager must draw the outline, frame, and background of the window. When closed, the window and its frame must be erased (i.e. redrawn in the screen's background color and pattern). "Moving" a window (changing its location on the screen) is essentially the same as closing and re-opening it.

A Window Manager can only be created and destroyed by a Console Manager, which is responsible for arranging windows on the screen, resolving overlaps, etc. When a Window Manager is created, it waits for an "initialize" message, initializes itself, returns an "I'm here" message to the process which sent it the "initialize" message, then waits for further messages. It does not send any messages to the Output Manager until it has received all of the following: its dimensions (exclusive of frame), the outline line-type, size and color, background color, location on the screen, a clipping polygon, scaling factors, and framing parameters. A Window Manager also has an "owner", which is a particular process which will handle commands (through the Console Manager, which always has prime control) within the window.

Any of the above parameters can be changed at any time. In general, changing any parameter (other than the owner) causes the window to be redrawn on the screen.

A "frame", which may consist of four components (called "bars"), one along each edge of the window, may be placed around the given window. The bars are designated top, bottom, left, and right. They can be any combination of simple line segment, title bar, scroll bar, menu bar, and palette bar. These are supplied to the message as four separate lists (in four separate messages) of standard picture elements, whcih can be changed at any time by sending a new message referencing the bar. The origin of each bar is [0,0] relative to the upper left corner of the window.

The Console Manager may query a Window Manager for any of its parameters, to which it responds with messages identical to the ones it originally received. It can also be asked whether a given absolute cursor position is inside its window (i.e. inside the current clipping polygon) or its frame, and for the cursor coordinates relative to the origin of the window or any edge of the frame.

A Window Manager is tightly coupled to its creator (a Console Manager), Picture Manager, and Output Manager; i.e. they communicate with each other using process identifiers (PID's). Consequently, a Window Manager must inform its Picture Manager when it exits, and it expects the Picture Manager to do the same.

Once the Window Manager knows the picture it is accessing and the dimensions of its window (or any time either of these changes), it requests the Picture Manager to send it all picture elements which completely or partially lie within the window. It also asks it to notify it of chanages which will affect the displayed portion of the picture. The Picture Manager will send "draw" messages to the Window Manager (at any time) to satisfy these requests.

The Window Manager performs gross clipping on all picture elements it receives, i.e. it just determines whether each element could appear inside the current clipping polygon (which may be smaller than the window at any given moment, if other windows overlap this one).

A Window Manager can be told to "freeze" (stop updating) its display and to "unfreeze" it. It can also be asked to redraw any given rectangular sub-area of the picture it is displaying.

Window Managers deal strictly in virtual pixels and have no knowledge about the physical characteristics of the screen to which they are writing. Consequently, a window's size and location are specified in virtual pixels, implying a conversion from real pixels if these are different.

Print Manager—There is one per "output subsystem", i.e. per pool of output devices. The Print Manager coordinates output to hard-copy devices (i.e. to their Output Managers). It provides a comprehensive queueing service for files that need to be printed. It can also perform some minimal formatting of text (justification, automatic page numbering, headers, footers, etc.)

The Print Manager knows about the following processes: Output Managers in the same context, and a Picture Manager in the same context. The following processes know about the Print Manager: any one that wants to.

One Print Manager is created automatically, at start-up time, in each Print context. It is expected to accept general requests for hard-copy output and pass them on, one message (usually corresponding to one "line" of output) at a time, to the appropriate Output Manager. It can also accept requests which refer to files (i.e. to File Manager processes). Each such message, known as a "spool" request, also contains a priority, the number of copies desired, specific output device requirements (if any) and special form requirements (if any).

Based on these parameters, as well as the size of the file, the amount of time the request has been waiting, and the availability of output devices, the Print Manager maintains an ordered queue of outstanding requests. It dequeues them one at a time, select an Output Manager, and builds a picture (using a Picture Manager). It then requests (from the Picture Manager) and "prints" (plots, etc.) one "page" at a time until the entire file has been printed.

The Print Manager recognizes specially marked ("tagged") picture elements which define headers, footers, foot-notes, and page formatting parameters (such as "page break", "set page number", etc.).

HUMAN INTERFACE—RELATIONSHIPS BETWEEN COMPONENTS

The eight Human Interface components together provide all of the services required to support a minimal human interface. The relationships between them are illustrated in FIG. 9, which shows at least one instance of each component. The components represented by circles **301**, **302**, **307**, **312**, **315**, and **317**-**320** are generally always present and active, while the other components are created as needed and exit when they have finished their specific funtions. FIG. **9** is divided into two main contexts: "Console" **350** and "Print" **351**.

Cursor **314** and Input **311** are examples of processes whose primary function is to store data. "Cursor"'s purpose is to keep track of the current cursor position on the screen and all parameters (such as the symbols defining different cursors) pertinent to the cursor. One cursor process is created by the Console Manager for each Output Manager when it is initialized. The Output Manager is responsible for updating the cursor data, although "Cursor" may be queried by anyone. "Input" keeps track of the current input state, such

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

as "select key is being held down", "keyboard locked", etc. One input process is created by each Console Manager. The console's input message updates the process; any other process may query it.

The Human Interface is structured as a collection of subsystems, implemented as contexts, each of which is responsible for one broad area of the interface. There are two major contexts accessible from outside the Human Interface: "Console" and "Print" They handle all screen/keyboard interaction and all hard-copy output, respectively. These contexts are not necessarily unique. There may be one or more instances of each in the system, with possibly several on the same cell. Within each, there may be several levels of nested contexts.

The possible interaction between various Human Interface components will now be described.

Console Manager/Other Contexts—Processes of other contexts may send requests for console services or notification of relevant events directly to the Console Manager(s). The Console Manager routes messages to the appropriate service. It also notifies (via a "status" message) the current owner of a window whenever an object in its window has been selected. Similarly, it sends a message to an application when a user requests that application in a particular window.

Console Manager/Input Manager—The Console Manager initializes the Input Manager and usually assigns a particular Output Manager to it. The Input Manager always sends all input (one character, one key, one cursor movement, etc. at a time) directly to the Console Manager. It may also send "status" messages, either in response to a "download", "initialize", or "terminate" request, or any time an anomaly arises.

Console Manager/Output Manager—The Console Manager displays information on its "prime" output device during system start-up and shut-down without using pictures and windows. It therefore sends picture elements directly to an Output Manager. The Console Manager is also responsible for moving the cursor on the screen while the system is running, if applicable. The Console Manager (or any other Human Interface manager, such as an "editor") may change the current cursor to any displayable symbol. Output Managers will send "status" messages to the Console Manager any time an anomaly arises.

Console Manager/Picture Manager—The Console Manager creates Picture Managers on demand and tells each of them the name of a file which contains picture elements, if applicable. A Picture Manager can also accept requests from the Console Manager (or anyone else) to add elements to a picture individually, delete elements, copy them, move them, modify their attributes, or transform them. It can be queried for the value of an element at (or close to) a given location within its picture. The Console Manager will tell a Picture Manager to erase its picture and exit when it is no longer needed. A Picture Manager usually sends "status" messages to the Console Manager whenever anything unusual (e.g., an error) occurs.

Console Manager/Window Manager—The Console Manager creates Window Managers on demand. Each Window Manager is told its size, the PID of an Output Manager, the coordinates (on the screen) of its upper left outside corner, the characteristics of its frame, the PID of a particular Picture Manager, the coordinates of the first element from which to start displaying the picture, and the name of the process which "owns" the window. While a window is active, it can be requested to re-display the same picture starting at a different element or to display a completely different picture.

The coordinates of the window itself may be changed, causing it to move on the screen, or it may be told to change its size, frame, or owner. A Window Manager can be told to "clip" the picture elements in its display along the edges of a given polygon (the default polygon is the inside edge of the window's frame). It can also be queried for the element corresponding to a given coordinate. The Console Manager will tell a Window Manager to "close" (erase) its window and exit when it is no longer needed. A Window Manager sends "status" messages to the Console Manager to indicate success or failure of a request.

Console Manager/Dialog Manager—The Dialog Manager accepts requests to load and/or dynamically create "pictures" which represent menus, prompts, error messages, etc. In the case of interactive pictures (such as menus), it also interprets the response for the Console Manager. Other processes may also use the Dialog Manager through the Console Manager.

Console Manager/Prime Manager—Console Managers generally send "spool" requests to Print Managers to get hard-copies of screens or pictures. An active picture must first be copied to a file. The Print Manager returns a "status" message when the request is complete or if it fails.

Window Manager/Picture Manager—A Window Manager requests lists of one or more picture elements from the relevant Picture Manager, specified by the coordinates of a rectangular "viewport" in the picture. It can also request the Picture Manager to automatically send changes (new, modified, or erased elements), or just notification of changes, to it. The Picture Manager sends "status" messages to notify the Window Manager of changes or errors.

Window Manager/Output Manager—A Window Manager sends lists of picture elements to its Output Manager, prefixed by the coordinates of a polygon by which the Output Manager is to "clip" the pixels of the elements as it draws them. A given list of picture elements can also be scaled by a given factor in any of its dimensions. The Output Manager returns a "status" message when a request fails.

Input Manager/Output Manager—The Input Manager sends all cursor movement inputs to a pre-assigned Output Manager (if any), as well as to the Console Manager. This assignment can be changed dynamically.

Print Manager/Other Processes—The Print Manager accepts requests to "spool" a file or to "print" one or more picture elements. It sends a "status" message at the completion of the request or if the request cannot be carried out. The status of a queued request can also be queried or changed at any time.

Print Manager/File Manager—The Print Manager reads picture elements from a File Manager (whose name was sent to it via a "spool" request). It may send a request to "delete" the file back to the File Manager after it has finished printing the picture.

Print Manager/Picture Manager—A Print Manager creates a Picture Manager for each spooled picture that it is currently printing, giving it the name of the relevant file. It then requests "pages" of the picture (depending upon the characteristics of the output device) one at a time. Finally, it tells the Picture Manager to go away.

Print Manager/Output Manager—The Print Manager sends picture elements to an Output Manager. The Output Manager sends a "status" message when the request completes or fails or when an anomaly arises on the printer.

Draw Manager/Other Processes—The Draw Manager accepts lists of elements prefixed by explicit pixel param-

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

eters (density, scaling factor, etc.). It returns a single message containing a list of bit-map ("symbol") elements of the drawn result for each message it receives.

HUMAN INTERFACE—SERVICE

A Human Interface service is accessed by sending a request message to the closest (i.e. the "next") Human Interface manager, or directly to a specific Console Manager. This establishes a "connection" to an existing Human Interface resource or creates a new one. Subsequent requests must be made directly to the resource, using the connector returned from the initial request, until the connection is broken. The Human Interface manager is distributed and thus spans the entire virtual machine. Resources are associated with specific nodes.

A picture may be any size, often larger than any physical screen or window. A window may only be as large as the screen on which it appears. There may be any number of windows simultaneously displaying pictures on a single screen. Updating a picture which is mapped to a window causes the screen display to be updated automatically. Several windows may be mapped to the same picture concurrently—at different coordinates.

The input model provided by the Human Interface consists of two levels of "virtual devices" The lower level supports "position", "character", "action", and "function key" devices associated with a particular window. These are supported consistently regardless of the actual devices connected to the system.

An optional higher level consists of a "dialog service", which adds "icons", "menus", "prompts", "values", and "information boxes" to the repertoire of device-independent interaction. Input is usually event-driven (via messages) but may also be sampled or explicitly requested.

All dimensions are in terms of "virtual pixels" A virtual pixel is a unit of measurement which is symmetrical in both dimensions. It has no particular size. Its sole purpose is to define the spatial relationships between picture elements. Actual sizes are determined by the output device to which the picture is directed, if and when it is displayed. One virtual pixel may translate to any multiple, including fractions, of a real pixel.

Using the core Human Interface services generally involves: creating a picture (or accessing a predefined picture); creating a window on a particular screen and connecting the picture to it; updating the picture (drawing new elements, moving or erasing old ones, etc.) to reflect changes in the application (e.g. new data); if the application is interactive, repeatedly accepting input from the window and acting accordingly; and deleting the picture and/or window when done.

Creating a new resource is done with an appropriate "create" message, directed to the appropriate resource manager (i.e. the Human Interface manager or Console Manager). Numerous options are available when a resource, particularly a window, is created. For example, a typical application may want to be notified when a specific key is pressed. Pop-up and pull-down menus, and function keys, may also be defined for a window.

All input from the Human Interface is sent by means of the "click" message. The intent of this message is to allow the application program to be as independent of the external input as possible. Consequently, a "click" generated by a pop-up menu looks very much like that generated by pressing a function key or selecting an icon. Event-driven input is initiated by a user interacting with an external device, such as a keyboard or mouse. In this case, the "click" is sent asynchronously, and multiple events are queued.

A program may also explicitly request input, using a menu, prompt, etc., in which case the "click" is sent only when the request is satisfied. A third method of input, which doesn't directly involve the user, is to query the current state of a virtual input device (e.g., the current cursor position).

A "click" message is associated with a particular window (and by implication usually with a particular picture), or with a dialog "metaphor", thus reflecting the two levels of the input model.

Since the visual aspect of the Human Interface is separated from the application aspect, a later redesign of a window, menu, icon, etc. has little or no effect upon existing applications.

HUMAN INTERFACE—DETAILED DESCRIPTION

Connectors

In general, all interaction with a Human Interface resource (console, window, picture, or virtual terminal) must be through a connector to that resource. Connectors to consoles can only be obtained from the Human Interface manager. Connectors to the other resources are available through the Human Interface manager, or through the Console Manager in which the desired resource resides. Requests must specify the path-name of the resource as follows:

[<console_name>][/<screen_name>][/<window_or_picture_name>]

That is, the name of the console, optionally followed by a slash and the name of the screen, optionally followed by a slash and the name of a window, picture, or terminal. The console name may be omitted only if the message is sent directly to the desired console manager. If the screen name is omitted, the first screen configured on the given console is assumed. The window name must be specified if one of those resources is being connected.

Connection Requests

The "create" and "open" requests can be addressed to the "next" Human Interface context ("HI") or to a specific console connector or to the "next" context named "Console". If sent to "HI", a full path-name (the name parameter) must be given; otherwise, only the name of the desired resource is required (e.g., at a minimum, just the name of the window or picture).

If a picture manager process is created locally by an application, for private use, an "init" message—with the same contents as "create" or "open" must be sent directly to the picture process. The response will be "done" or "failed".

The following are the various Connection Requests and the types of information which may be associated with each:

CREATE is used to create a new picture resource, a new window resource, or a new virtual terminal resource.

When used to create a new picture resource, it may contain information about the resource type (i.e. a "picture"); the path-name of the picture; the size; the background color; the highlighting method; the maximum number of elements; the maximum element size; and the path-name of a library picture from which other elements may be copied.

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

When used to create a new window resource, it may contain information about the resource type (i.e. a "window"); the path-name of the window; the window's title; the window's position on the screen; the size of the window; the color, width, fill color between the outline and the pane, and the style of the main window outline; the color and width of the pane outline; a mapping of part of a picture into the window; a modification notation; a special character notation; various options; a "when" parameter requesting notification of various specified actions on/within the window; a title bar; a palette bar; vertical and horizontal scroll bars; a general use bar; and a corner box.

When used to create a new virtual terminal, it may contain information about the resource type (i.e. a "terminal"); the path-name of the terminal; the title of the terminal's window; various options; the terminal's position on the screen; the size of the terminal (i.e. number of lines and columns in the window); the maximum height and width of the virtual screen; the color the text inside the window; tab information; emulator process information; connector information to an existing window; window frame color; a list of menu items; and alternative format information.

OPEN is used to connect to a Human Interface service or to an existing Human Interface resource. When used to connect to a Human Interface service, it may contain information about the service type; and the name of the particular instance of the service. This resource must be sent to the Human Interface context.

When used to connect to an existing Human Interface resource, it may contain information about the path-name of the resource; the type of resource (e.g. picture, window, or terminal); and the name of the file (for pictures only) from which to load the picture. This request can be sent to a Human Interface manager or a console manager; alternatively the same message with message I.D. "init" specifying a file can be sent directly to a privately owned picture manager.

DELETE is used to remove an existing Human Interface resource from the system, and it may contain information specifying a connection to the resource; the type of resource; and whether, for a window, the corresponding picture is to be deleted at the same time.

CLOSE is used to break a connection to a Human Interface resource, and it may contain information specifying a connection to the resource; and the type of resource.

WHO? is used to request a list of signed-on users, and it may contain a user identification string.

QUERY is used to get the status of a service or resource, and it may contain information about the resource type; the name of the service or resource; a connector to a resource; and information concerning various options.

The following are the various Connection Responses and the types of information which may be associated with each:

CONNECT provides a connection to a Human Interface resource, and it contains information concerning the originator (i.e. the Human Interface or the console); the resource type; the original request message identifier; the name of the resource; and a connector to the resource.

USER contains the names of zero or more currently signed-on users and their locations, and it contains a connector to a console manager followed by the name of the user signed on at that console.

Console Requests

The main purpose of the console is to coordinate the activities of the windows, pictures, and dialog associated with it. Any of the CREATE, OPEN, DELETE, and CLOSE connection requests listed above, except those relating to the consoles, can be sent directly to a known console manager, rather than to the Human Interface manager (which always searches for the console by name). Subsequently, some characteristics of a window, such as its size, can be changed dynamically through the console manager. The current "user" of the console can be changed. And the console can be queried for its current status (or that of any of its resources).

The following are the various Console Requests and the types of information which may be associated with each:

USER is used to change the currently signed-on user, and it contains a user identification string.

CHANGE is used to change the size and other conditions of a window, and it may contain information about a connector to a window or a terminal; new height and width (in virtual pixels); increment to height and width; row and column position; various options; a connector to a new owner process; and whether the window should be the current active window on the screen.

CURSOR is used to move the screen cursor, and it contains position information as to row and column.

QUERY is used to get the current status of the console or one of its resources, and it contains information in the form of a connector to the resource; and various query options (e.g. list all screens, all pictures, or all windows).

BAND starts/stops the rubber-banding function and dragging function, and it contains information about the position of a point in the picture from which to start the operation; the end point of the figure which is to be dragged; the type of operation (e.g. line, rectangle, circle, or ellipse); the color; and the type of line (e.g. solid). In rubber-banding the drawn figure changes in size as the cursor is moved. In dragging the figure moves with the cursor.

The following are the various Console Responses and the types of information which may be associated with each:

STATUS describes the current state of a console, and it may contain information about a connector to the console; the originator; the name of the console; current cursor position; current metaphor size; scale of virtual pixels per centimeter, vertically and horizontally; number of colors supported; current user i.d. string; screen size and name; window connector and name; and picture connector, screen name, and window name.

Picture-Drawing

The picture is the fundamental building block in the Human Interface. It consists of a list of zero or more "picture elements", each of which is a device-independent abstraction of a displayable object (line, text, etc.). Each currently active picture is stored and maintained by a separate picture manager. "Drawing" a picture consists of sending picture manipulation messages to the picture manager.

A picture manager must first be initialized by a CREATE or OPEN request (or INIT, if the picture was created privately). CREATE sets the picture to empty, gives it a name, and defines the background. The OPEN request reads a predefined picture from a file and gives it a name. Either must be sent first before anything else is done. A subsequent OPEN reloads the picture from the file.

The basic request is to WRITE one or more elements. WRITE adds new elements to the end of the current list, thus

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

Preceding an item with "+" indicates that the item is currently "active" and causes a check mark to be displayed beside it whenever the menu is opened. Preceding an item with "–" indicates that the corresponding option is not currently available and cannot be selected.

An "arguments" string can be appended to the tag of an element in the menu. The string is passed "as is" to the application when the item is selected.

PROMPT

The greater part of a prompt picture comprises text which asks a question, often with some introductory preamble. One element, located anywhere in the picture, may represent a response area. This is generally a rectangular area into which a user can type the information requested by the prompt. This element must be tagged "RESP".

Two further elements, tagged "ENTER" and "CANCEL", display target text or symbols which are used to complete the prompt. When the "enter" element is selected by the user, the text typed in the response area is returned to the originator of the prompt.

If the "cancel" element is selected instead, the prompt is cancelled with a null response. The response element is optional. If omitted, the "enter" and "cancel" elements effectively correspond to "yes" or "no" responses. Typing a "carriage return" character will have the same effect as selecting "enter" The prompt is erased when any response is given, or by an explicit "cancel" request.

INFORMATION

An information picture comprises text (and possibly graphics) which describes something. One element, located anywhere in the picture, is usually tagged "DONE" When this element is selected, the information picture is erased from the display. If no such element is given, the process which requested the information to be displayed must send an explicit "cancel" request when it wants to get rid of it.

INPUT/OUTPUT DEVICE INDEPENDENCE

In the present invention all system interaction with the outside world is either through "virtual input" or "virtual output" devices. The system can accept any form of input or output device. The Human Interface is constructed using a well-defined set of "virtual devices" All Human Interface functions (windowing, picture—drawing, dialog management, etc.) use this set of devices exclusively.

These virtual input devices take the form of "keys" (typed textual input); "position" (screen coordinates); "actions" (Human Interface functions such as "open window", etc.); "functions" (user-defined actions); and "means" (pop-up lists of choices).

Virtual output devices produce device-independent output: text, lines, rectangles, polygons, circles, ellipses, discrete points, bit-mapped symbols, and bit-mapped arrays.

FIG. 12 shows how the console manager operates upon virtual input to generate virtual output. The lowest layer of HI software converts input from any "real" physical devices to the generic, virtual form, and it converts Human Interface output (in virtual form) to physical output.

FIG. 12 shows the central process of the Human Interface, the console manager 220, dealing with virtual input and producing virtual output. Virtual input passed through the virtual input manager 221 is processed directly by the console manager 220, while output is passed through two intermediate processes—(1) a picture manager 222, which manipulates device—independent graphical images, and (2) a window manager 224, which presents a subset (called a "view") of the overall picture to the virtual output manager 226.

Any number of physical devices can be connected to the Human Interface and can be removed or replaced dynamically, without disturbing the current state of the Human Interface or of any applications using the Human Interface. In other words, the Human Interface is independent of particular I/O devices, and the idiosyncracies of the devices are hidden from the Human Interface.

FIG. 13 represents a flowchart showing how virtual input is handled by the console manager. The virtual input may take any of several forms, such as a keystroke, cursor position, action, function key, menu, etc.

For example, regarding the operations beneath block 231, if the virtual input to the console manager is a keystroke, then the console manager checks to see whether the cursor is inside a window. If so, it checks to see whether it originated from a virtual terminal, and if not it checks to see whether an edit operation is taking place. If not, it updates the picture.

Regarding the operations beneath block 232, if the virtual input represents a cursor position, then the console manager checks to see whether the auto-highlight option has been enabled. If yes, it checks to see whether the cursor is on an element. If so it highlights that element.

Regarding the operations beneath block 233, the console manager uses any of the indicated actions to update a picture, update a window, or initiate dialog, as appropriate.

Regarding the operations beneath block 234, if the virtual input is from a function key, the console manager notifies the dialog manager.

Regarding the operations beneath block 235, if the virtual input represents a menu choice, the console manager checks to see whether the cursor is in a window. If not, it determines that it is on a user metaphor; if so, it requests a menu from the window. If the menu is defined, it notifies the owner of the window (or metaphor), activates a pop-up menu, gets a response, and sends the response to the window owner.

FIG. 14 represents a flowchart showing how virtual output is handled by the picture manager. The picture manager 240 accepts virtual output from the console manager and then, depending upon the type of operation, performs the requested function. For example, if the operation is a replace operation, the picture manager 240 replaces the old output with the new and sends the change(s) to the window manager. The window manager sends the change to the output manager, which in turn sends it to the real device.

DESCRIPTION OF SOURCE CODE LISTING

Program Listings A and B contain a "C" language implementation of the above-described concepts relating to input/output device independence. The following chart indicates where the relevant portions of the listing may be found.

Function	Lines Numbers in Program Listing A
Main-line; initialization; accept input	190–222
Determine type of input	486–521
Virtual key	523–631
Virtual position	633–661
Virtual action	663–702, 763–1200
Virtual function	704–723
Virtual menu	725–761
	Lines Numbers in Program Listing B
Main-line; initialization; start processing	125–141
Accept requests (virtual output); check for changes	161–203
Determine type of request	239–310

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

```
188  /* Console manager:  main-line */
189
190  PROCESS(Console)
191  {
192      NAME                     *name;
193      SCREEN                   *screen;
194      LIST                     *map_ptr;
195      SELECTION                *sel;
196      WINDOW                   *window;
197      MESSAGE                  *msg_ptr;
198      CONNS                    *conn_ptr;
199      register LIST            *map;
200      register MESSAGE         *msg;
201      register CONNS           *conn;
202      register short           go = YES;
203      long                     list_size = 0, *req = NULL;
204
205      Set_event_key("Console mgr.");
206      init_CM(&name,&screen,&map_ptr,&sel,&window,&msg_ptr,&conn_ptr);
207      map = map_ptr;
208      msg = msg_ptr;
209      conn = conn_ptr;
210      start_up(name,screen,conn);
211      while (go)
212      {
213          msg->buf = Get(0,&msg->sender,&msg->size);
214          if (!*(msg->buf+1))
215              input(screen,map,sel,window,msg,conn,*msg->buf);
216          else
217              request(name,screen,map,sel,msg,conn,msg->buf,msg->size);
218          highlight(map->active_map);
219          free_requests(msg->buf,msg->size,&req,&list_size);
220      }
221      Exit();
222  }
```

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS
INFORMATION, SUBJECT TO PROTECTIVE ORDER

```
223  free requests(msg,size,req,list_size)
224  register char     *msg, **req;
225  register long      size, *list_size;
226  {
227      register char    *temp, *next;
228
229      if (msg)
230      {
231          *(char**)msg = *req;
232          *req = msg;
233          *list_size += size;
234          if (!Any msg(NULL) || *list_size > 1000)
235              for (temp = *req, *req = NULL, *list_size = 0; temp; temp = next)
236              {
237                  next = *(char**)temp;
238                  Free(temp);
239              }
240      }
241  }
242
```

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS
INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019534

```
243
244  init CM(name,screen,map,sel,window,msg,conn)
245  register NAME        **name;
246  register SCREEN      **screen;
247  register LIST        **map;
248  register SELECTION   **sel;
249  WINDOW               **window;
250  MESSAGE              **msg;
251  CONNS                **conn;
252  {
253      *name = (NAME *) Alloc(sizeof(NAME),YES);
254      *screen = (SCREEN *) Alloc(sizeof(SCREEN),YES);
255      *map = (LIST *) Alloc(sizeof(LIST),YES);
256      *sel = (SELECTION *) Alloc(sizeof(SELECTION),YES);
257      *window = (WINDOW *) Alloc(sizeof(WINDOW),YES);
258      *msg = (MESSAGE *) Alloc(sizeof(MESSAGE),YES);
259      *conn = (CONNS *) Alloc(sizeof(CONNS),YES);
260      memset(*name,0,sizeof(NAME));
261      strcpy(*name,"name:");
262      memset(*screen,0,sizeof(SCREEN));
263      strcpy(*screen,"screen:");
264      memset(*map,0,sizeof(LIST));
265      strcpy(*map,"list:");
266      memset(*sel,0,sizeof(SELECTION));
267      strcpy(*sel,"selection:");
268      memset(*window,0,sizeof(WINDOW));
269      strcpy(*window,"window:");
270      memset(*msg,0,sizeof(MESSAGE));
271      strcpy(*msg,"message:");
272      memset(*conn,0,sizeof(CONNS));
273      strcpy(*conn,"conns:");
274      (*map)->pool = (MAPNODE *) Alloc(POOL SIZE*sizeof(MAPNODE),YES);
275      memset((*map)->pool,0,POOL_SIZE*sizeof(MAPNODE));
276  }
```

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS
INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019535

```
277
278 start_up(name,screen,conn)
279 register NAME    *name;
280 register SCREEN *screen;
281 register CONNS  *conn;
282 {
283     register char     *msg;
284     CONNECTOR         config;
285     short             *p;
286     long              size;
287
288     while ((msg = Get(0,&conn->owner,&size)) && strcmp(msg,"init"))
289     {
290         reply_status(msg,msg,"not ready",0);
291         Free(msg);
292     }
293     strcpy(name->console,Find_triple(msg,"name",size,none,2,NULL));
294     conn->self = *(CONNECTOR *) Find_triple(msg,"self",size,none,4,NULL);
295     Free(msg);
296     if (config.pid = NewProc("CMconfig","//processes/CMconfig",YES,-1))
297     {
298         Put(DIRECT,config.pid,Newmsg(32,"I",NULL));
299         while (!Any_msg(config.pid))
300             if (Any_msg(conn->owner.pid))
301                 Forward(DIRECT,config.pid,Get(conn->owner.pid,0,0));
302             else
303                 Free(Call(NEXT,"Clock",
304                     Newmsg(64,"set","aftr=#5s",0,0,0,5,0),0,0));
305         msg = Get(config.pid,0,&size);
306         conn->input = *(CONNECTOR*) Find_triple(msg,"inp ",size,none,4,NULL);
307         conn->output = *(CONNECTOR*)Find_triple(msg,"outp",size,none,4,NULL);
308         conn->dialogue = *(CONNECTOR*) Find_triple(msg,"dial",size,none,4,NULL);
309         Free(msg);
310         if (msg = Call(DIRECT,conn->output.pid,Newmsg(32,"query",NULL),0,&size))
311         {
312             p = (short *) Find_triple(msg,"scrn",size,none,4,NULL);
313             screen->meta_ht = screen->height = *p++;
314             screen->meta_wd = screen->width = *p;
315             screen->char_gen = screen->char_align =
316                 (char) Find_triple(msg,"char",size,NO,0,NULL);
317             screen->colors = *(short*) Find_triple(msg,"clrs",size,none,2,NULL);
318             screen->bit_map = (char) Find_triple(msg,"bmap",size,NO,0,NULL);
319             screen->fonts = (char) Find_triple(msg,"font",size,NO,0,NULL);
320             Free(msg);
321         }
322         else
323             Note("'query' to output mgr. failed",msg);
324         Put(DIRECT,conn->owner.pid,
325             Newmsg(128,"ready","serv=#S; name=#S","console",name->console));
326     }
327 }
```

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER

```
328
329  request(name,screen,map,sel,msg,conn,buf,size)
330  register NAME          *name;
331  SCREEN                 *screen;
332  register LIST          *map;
333  SELECTION              *sel;
334  register MESSAGE       *msg;
335  register CONNS         *conn;
336  register long          buf, size;
337  {
338       if (!strcmp(buf,"create"))
339            Create_resource(screen,map,buf,size,&conn->output,&msg->sender);
340       else if (!strcmp(buf,"write"))
341            element_selected(map,sel,msg);
342       else if (!strcmp(buf,"delete"))
343            Delete_resource(map,msg,conn,sel);
344       else if (!strcmp(buf,"Meta"))
345            Metaphor(screen,map,buf,size,&conn->output,&conn->dialogue);
346       else if (!strcmp(buf,"user"))
347            Set_user(name,buf,size);
348       else if (!strcmp(buf,"resource"))
349            ;
350       else if (!strcmp(buf,"query"))
351            Query(name,screen,map,msg,conn);
352       else if (!strcmp(buf,"change"))
353            Change(screen,map,msg);
354       else if (!strcmp(buf,"remapped"))
355            remap(&msg->sender,NULL,Find_triple(buf,"conn",0,0,8,0),sel,map);
356       else if (!strcmp(buf,"failed"))
357            Status(buf,size);
358       else if (!strcmp(buf,"done") || !strcmp(buf,"status"))
359            ;
360       else if (conn->dialogue.pid)
361       {
362            buf = (long) Realloc(buf,size+20,YES);
363            Append_triple(buf,"Cpos",4,&screen->row);
364            Forward(DIRECT,conn->dialogue.pid,buf);
365            msg->buf = NULL;
366       }
367       else
368            reply_status(buf,buf,"unknown msg id",0);
369  }
```

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS
INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019537

```
370
371 Query(name,screen,map,msg,conn)
372 NAME        *name;
373 SCREEN      *screen;
374 LIST        *map;
375 MESSAGE     *msg;
376 CONNS       *conn;
377 {
378     static char          def_res[] = "console";
379     register char        *window_name, *resource, *p;
380     register MAPNODE     *node = NULL;
381     CONNECTOR            *res;
382
383     resource = Find_triple(msg->buf,"res ",msg->size,def_res,2,NULL);
384     if (!strcmp(resource,"console"))
385         Reply(msg->buf,Newmsg(500,"console",
386             "name=#S; user=#S; clrs=#s; conn=#C; orig=#S",
387             name->console,name->user,screen->colors,&conn->self,"console"));
388     else
389     {
390         if (window_name = Find_triple(msg->buf,"name",msg->size,NULL,2,NULL))
391         {
392             if (!(p = strrchr(window_name,'/')))
393                 p = window_name;
394             for (node = map->first;
395                 node && strcmp(p,node->name); node = node->nxt) ;
396         }
397         else if (res = (CONNECTOR*) Find_triple(msg->buf,"conn",0,NULL,1,NULL))
398             for (node = map->first; node && node->window.pid != res->pid
399                 && node->picture.pid != res->pid
400                 && node->terminal.pid != res->pid; node = node->nxt) ;
401         else
402             reply_status(msg->buf,"-query","missing name/connector",0);
403         if (node)
404         {
405             if (!strcmp(resource,"window"))
406                 Forward(DIRECT,node->window.pid,msg->buf);
407             else if (!strcmp(resource,"terminal") && node->terminal.pid)
408                 Forward(DIRECT,node->terminal.pid,msg->buf);
409             else if (!strcmp(resource,"picture") && node->picture.pid)
410                 Forward(DIRECT,node->picture.pid,msg->buf);
411             else
412                 Free(msg->buf);
413             msg->buf = NULL;
414         }
415     }
416 }
```

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS
INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM 24063 0019538

```
417
418 Create_resource(screen,map,buf,size,output,sender)
419 SCREEN          *screen;
420 LIST            *map;
421 CONNECTOR       *output, *sender;
422 register long   buf, size;
423 {
424     static char             def_res[] = "window";
425     register char           *resource, *p;
426     register MAPNODE        *node = NULL;
427     register CONNECTOR      *conn = NULL;
428     CONNECTOR               picture;
429
430     resource = Find_triple(buf,"res ",size,def_res,2,NULL);
431     if (!strcmp(resource,"window")
432         && (node = create_window(screen,map,output,"Window",buf,size)))
433         {
434             conn = &node->window;
435             node->owner = *sender;
436         }
437     else if (!strcmp(resource,"terminal") && (node =
438         create_terminal(screen,map,output,buf,size,sender)))
439             conn = &node->terminal;
440     else if (!strcmp(resource,"picture"))
441         if (picture.pid = NewProc("Picture","//processes/picture",YES,-1))
442         {
443             p = Alloc(size,YES);
444             memcpy(p,buf,size);
445             Free(Call(DIRECT,picture.pid,p,0,0));
446             conn = &picture;
447         }
448     if (conn)
449         Reply(buf,Newmsg(200,"connect","conn=#C; orig=#S; req=#S; res=#S",
450             conn,"console","create",resource));
451     else
452         reply_status(buf,"-create","unknown resource type",0);
453     activate(node);
454 }
455
456 Delete_resource(map,msg,conn,sel)
457 LIST                *map;
458 register MESSAGE    *msg;
459 register CONNS      *conn;
460 SELECTION           *sel;
461 {
462     register MAPNODE        *node, *temp;
463     CONNECTOR               *resource;
464
```

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS
INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019539

```
465        if (resource=(CONNECTOR*)Find_triple(msg->buf,"conn",msg->size,NULL,8,NULL))
466            if (!strcmp(Find_triple(msg->buf,"res ",0,NULL,2,NULL),"picture"))
467            {
468                Put(DIRECT,resource->pid,Newmsg(32,"quit",NULL));
469                remap(&msg->sender,NULL,NULL,sel,map);
470            }
471            else
472            {
473                temp = map->active;
474                for (node = map->first;
475                     node && node->window.pid != resource->pid
476                     && node->picture.pid != resource->pid
477                     && node->terminal.pid != resource->pid; node = node->nxt) ;
478                if (node)
479                    close_window(node,map,sel,conn);
480                if (Find_triple(msg->buf,"rply",msg->size,NO,0,NULL))
481                    reply_status(msg->buf,"+delete","resource deleted",cX_DELETED);
482                map->active = temp;
483            }
484    }
485
486    input(screen,map,sel,window,msg,conn,msgid)
487    SCREEN              *screen;
488    LIST                *map;
489    SELECTION           *sel;
490    register WINDOW     *window;
491    register MESSAGE    *msg;
492    CONNS               *conn;
493    register char       msgid;
494    {
495        register char       code;
496        register short      *pos;
497        register MAPNODE    *node;
498
499        pos = (short *) Find_once(msg->buf,"pos ",msg->size,none,4,NULL);
500        code = *Find_triple(msg->buf,"\0\0\0\0",msg->size,none,1,NULL);
501        node = map->active;
502        if (msgid == 'K' && node)
503            key_input(node,window,msg,code);
504        else if (msgid == 'F' && node)
505            function_key(node,code,&conn->dialogue);
506        else
507        {
508            node = find_window(map,window,*pos,*(pos+1));
509            if (msgid == 'P')
510            {
511                if (node && window->area == 'I')
512                    position(node,window);
513                screen->row = *pos;
514                screen->col = *(pos+1);
```

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS
INFORMATION, SUBJECT TO PROTECTIVE ORDER

MOTM_24063_0019540

What is claimed is:

1. A virtual input interface in a data processing system, said interface comprising:

means for accepting input from at least one physical device and for converting said physical device input into virtual input, said means comprising a virtual input manager process responsive to said at least one physical input device for generating a picture, said picture comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element; and

means responsive to said virtual input for performing processing operations upon said virtual input, said means comprising a console manager process for performing processing operations on said one or more picture elements.

2. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of keystrokes.

3. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of cursor position.

4. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of system-defined actions.

5. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of user-defined functions.

6. The virtual input interface as recited in claim 1, wherein said input accepting means accepts input in the form of menu selections.

7. The virtual input interface as recited in claim 1, wherein said at least one physical device can be removed from said system without affecting the operation of the remainder of said system.

8. The virtual input interface as recited in claim 1, wherein at least one additional physical device can be added to said system without affecting the operation of the remainder of said system.

9. A virtual output interface in a data processing system, said interface comprising:

a source of virtual input, said virtual input comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element;

means for performing processing operations on said virtual input and for generating virtual output;

means for accepting said virtual output; and

means for converting said virtual output into at least one physical output suitable for use by at least one physical output device.

10. The virtual output interface as recited in claim 9, wherein said virtual input comprises a plurality of related picture elements and wherein said virtual output accepting means comprises a picture manager process for controlling said plurality of related picture elements.

11. The virtual output interface as recited in claim 10 and further comprising a display device, wherein said virtual output accepting means further comprises a window manager process for controlling the display of said plurality of related picture elements on said display device.

12. The virtual output interface as recited in claim 9, wherein said virtual output converting means comprises a virtual output manager process responsive to said one or more processed picture elements for coupling said one or more processed picture elements to said at least one physical output device.

13. The virtual output interface as recited in claim 9, wherein said at least one physical device can be removed from said system without affecting the operation of the remainder of said system.

14. The virtual output interface as recited in claim 9, wherein at least one additional physical device can be added to said system without affecting the operation of the remainder of said system.

15. In a data processing system, an interface between processes and data in said system and physical input and output devices coupled to said system, said interface comprising:

means responsive to one of said physical input devices for generating a picture, said picture comprising one or more picture elements, each picture element comprising a plurality of device-independent data structures in a predetermined, standard data format, at least one of said data structures comprising a plurality of different data fields each containing information describing said picture element;

means for performing processing operations on said one or more picture elements; and

means responsive to said one or more processed picture elements for coupling said one or more processed picture elements to one of said physical output devices.

16. The data processing system as recited in claim 15, wherein said one or more picture elements define a graphical object and at least one attribute thereof.

17. The data processing system as recited in claim 16, wherein one of said data fields describes the length of the associated picture element.

18. The data processing system as recited in claim 16, wherein one of said data fields identifies the particular type of the associated picture element.

19. The data processing system as recited in claim 16, wherein one of said data fields describes the position of the associated picture element relative to row and column coordinates on a picture of which said picture element forms a part.

20. The data processing system as recited in claim 16, wherein one of said data fields describes the size of the associated picture element.

21. The data processing system as recited in claim 16, wherein one of said data fields describes the color of the associated picture element.

22. The data processing system as recited in claim 15, wherein said means responsive to one of said physical input devices comprises a virtual input manager process.

23. The data processing system as recited in claim 15, wherein said means responsive to said one or more processed picture elements comprises a virtual output manager process.

* * * * *

CONTAINS MOTOROLA MOBILITY, INC. CONFIDENTIAL BUSINESS INFORMATION, SUBJECT TO PROTECTIVE ORDER