EXHIBIT 1.06

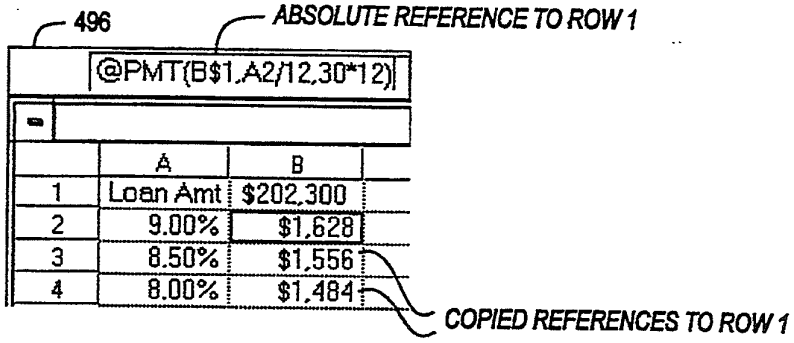


FIG. 4H

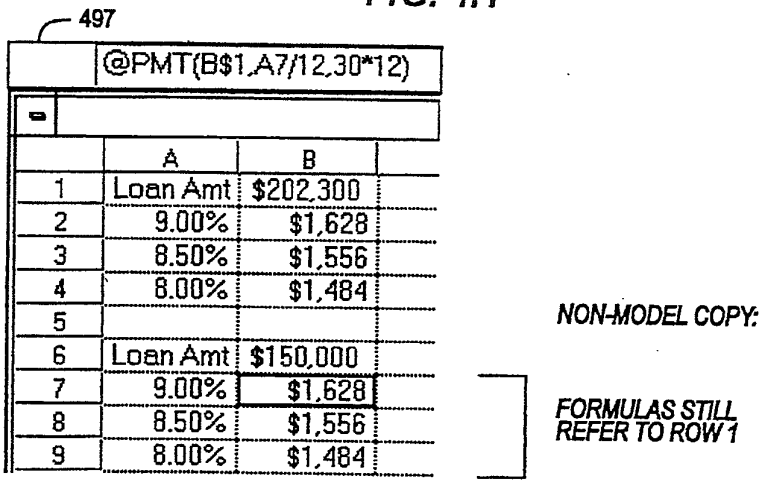


FIG. 4I

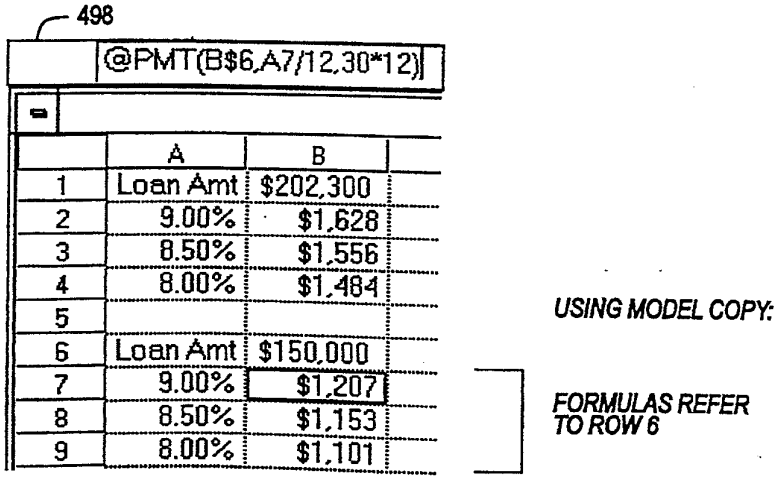


FIG. 4J

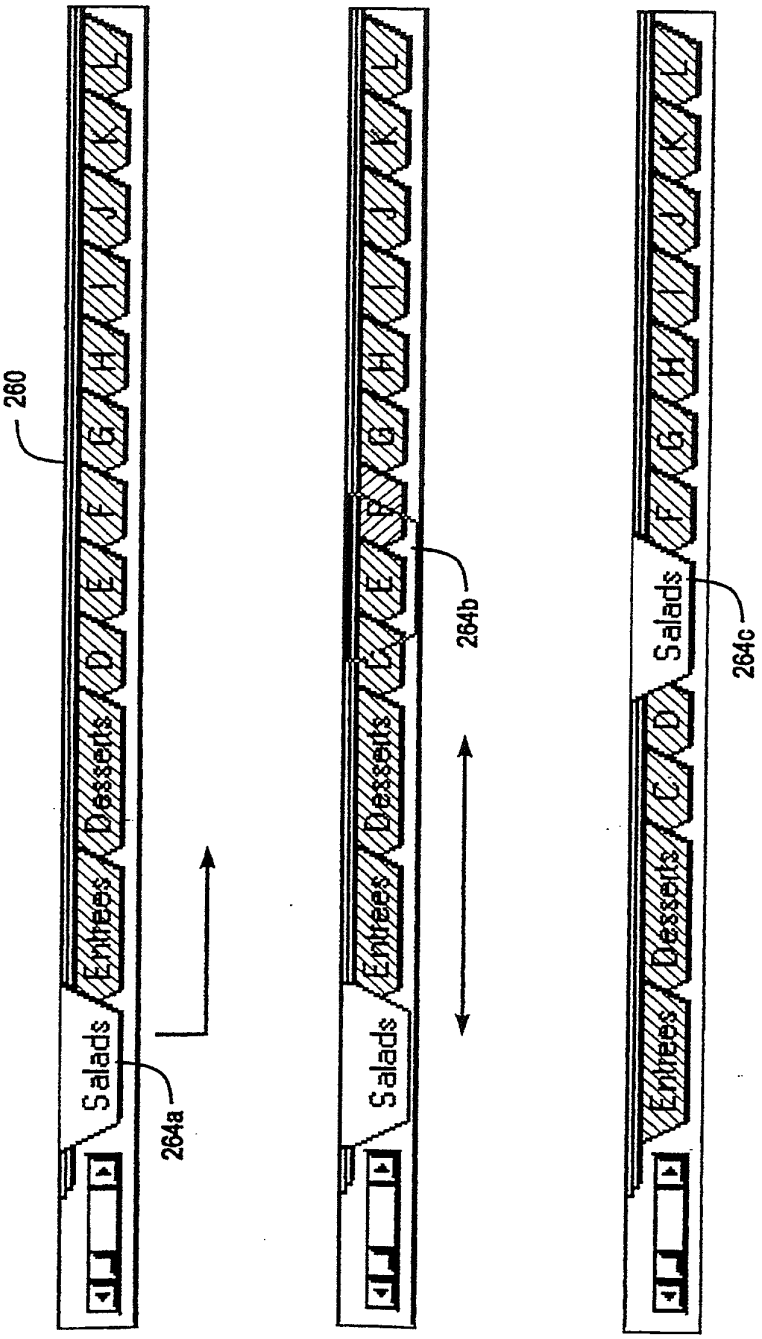
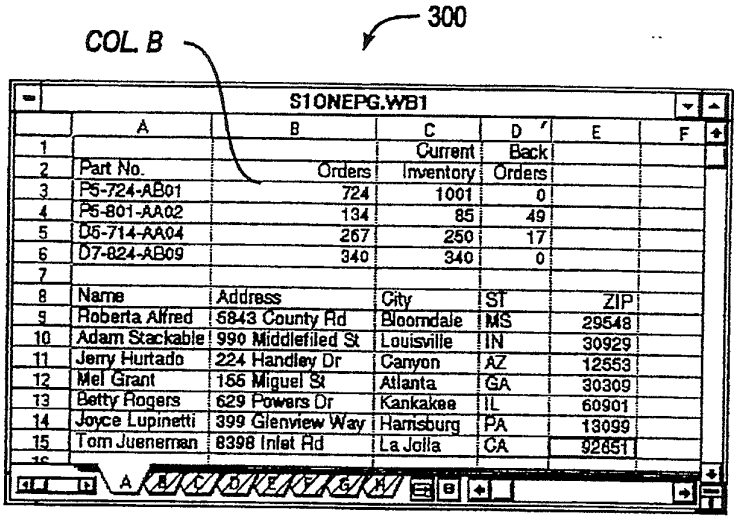
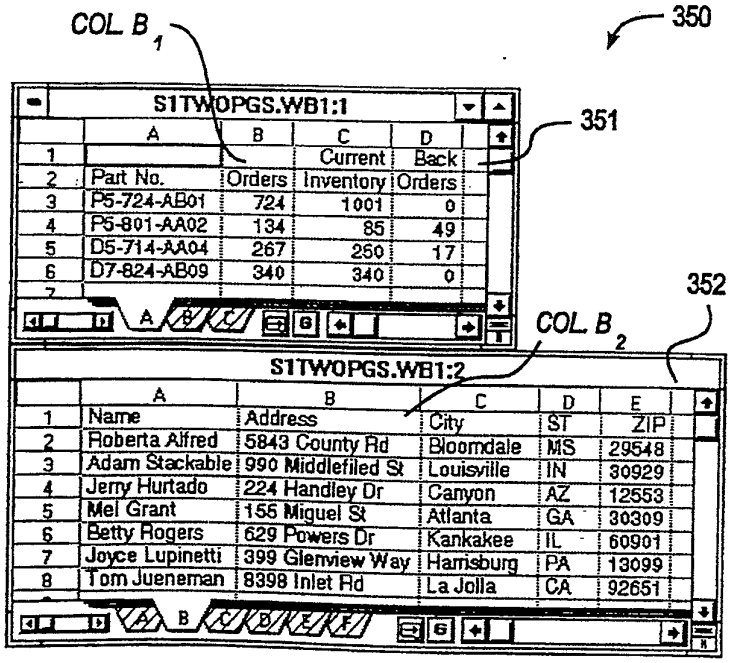


FIG. 4K



	A	B	C	D	E	F
1			Current	Back		
2	Part No.	Orders	Inventory	Orders		
3	P5-724-AB01	724	1001	0		
4	P5-801-AA02	134	85	49		
5	D5-714-AA04	267	250	17		
6	D7-824-AB09	340	340	0		
7						
8	Name	Address	City	ST	ZIP	
9	Roberta Alfred	5843 County Rd	Bloomdale	MS	29548	
10	Adam Stackable	990 Middlefiled St	Louisville	IN	30929	
11	Jerry Hurtado	224 Handley Dr	Canyon	AZ	12553	
12	Mel Grant	156 Miguel St	Atlanta	GA	30309	
13	Betty Rogers	629 Powers Dr	Kankakee	IL	60901	
14	Joyce Lupinetti	399 Glenview Way	Harrisburg	PA	13099	
15	Tom Jueneman	8398 Inlet Rd	La Jolla	CA	92651	

FIG. 4L



	A	B	C	D
1			Current	Back
2	Part No.	Orders	Inventory	Orders
3	P5-724-AB01	724	1001	0
4	P5-801-AA02	134	85	49
5	D5-714-AA04	267	250	17
6	D7-824-AB09	340	340	0

	A	B	C	D	E
1	Name	Address	City	ST	ZIP
2	Roberta Alfred	5843 County Rd	Bloomdale	MS	29548
3	Adam Stackable	990 Middlefiled St	Louisville	IN	30929
4	Jerry Hurtado	224 Handley Dr	Canyon	AZ	12553
5	Mel Grant	156 Miguel St	Atlanta	GA	30309
6	Betty Rogers	629 Powers Dr	Kankakee	IL	60901
7	Joyce Lupinetti	399 Glenview Way	Harrisburg	PA	13099
8	Tom Jueneman	8398 Inlet Rd	La Jolla	CA	92651

FIG. 4M

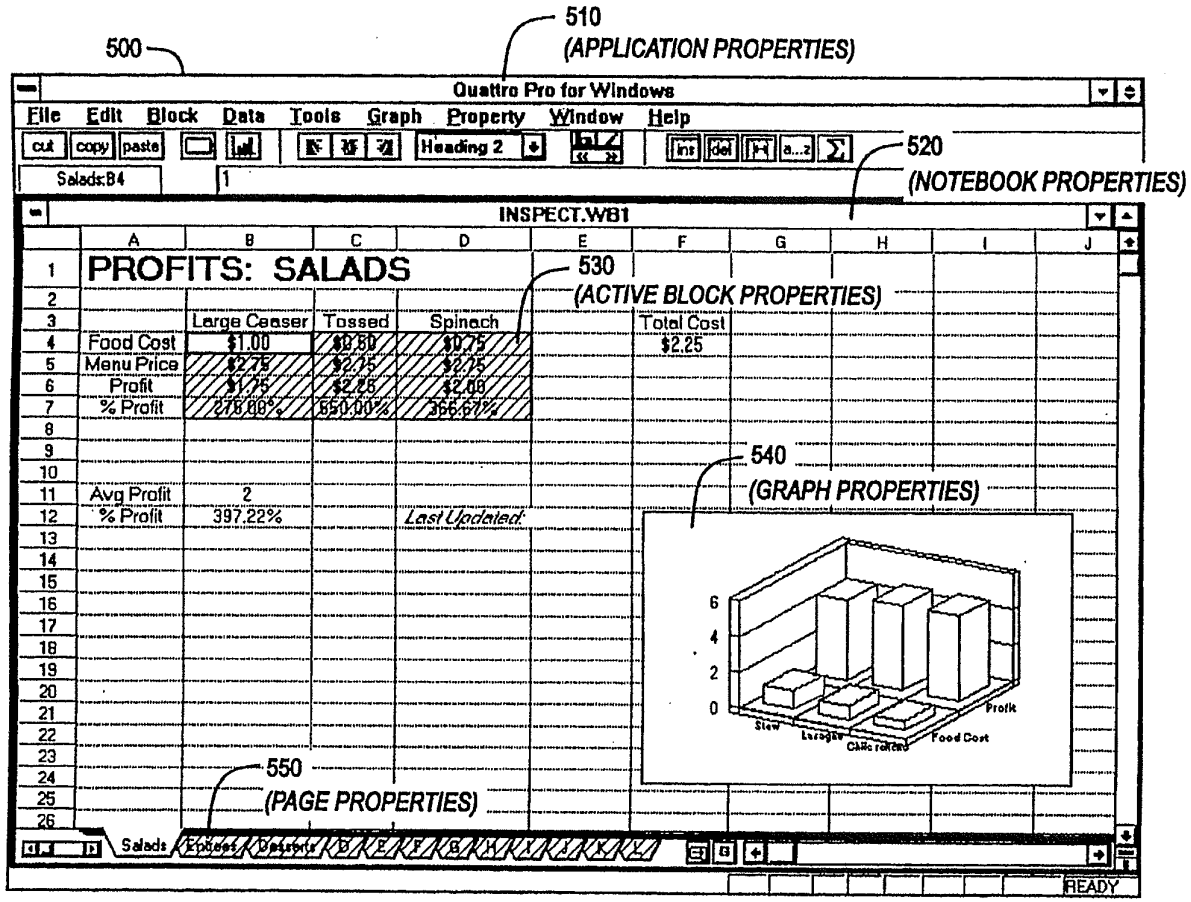

500
510
(APPLICATION PROPERTIES)
Quattro Pro for Windows
File Edit Block Data Tools Graph Property Window Help
cut copy paste
Heading 2
Salads:B4
520
(NOTEBOOK PROPERTIES)
INSPECT.WB1
PROFITS: SALADS
530
(ACTIVE BLOCK PROPERTIES)
Large Ceaser Tossed Spinach
Total Cost
Food Cost $1.00 $0.50 $0.75 $2.25
Menu Price $2.75 $2.75 $2.75
Profit $1.75 $2.25 $2.00
% Profit 276.00% 550.00% 366.67%
Avg Profit 2
% Profit 397.22%
Last Updated:
540
(GRAPH PROPERTIES)
6
4
2
0
Profit
Food Cost
550
(PAGE PROPERTIES)
Salads
READY

FIG. 5A

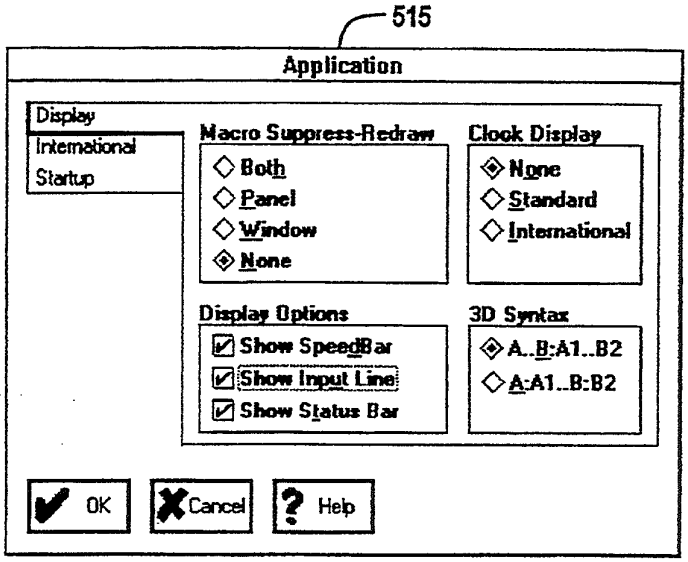


FIG. 5B

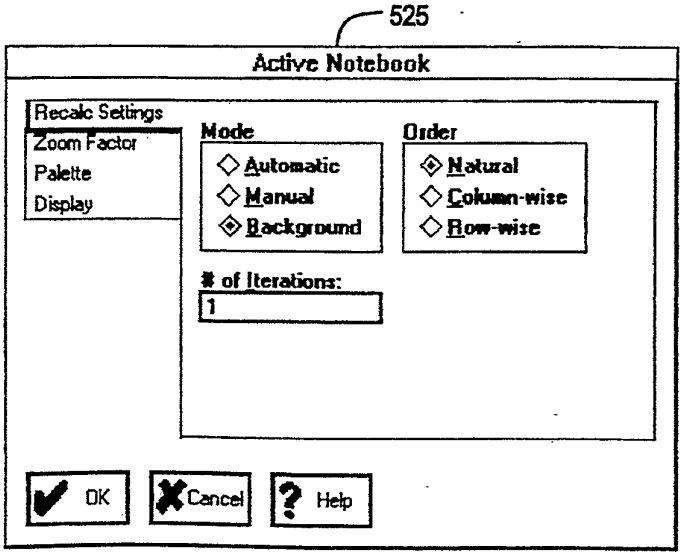


FIG. 5C

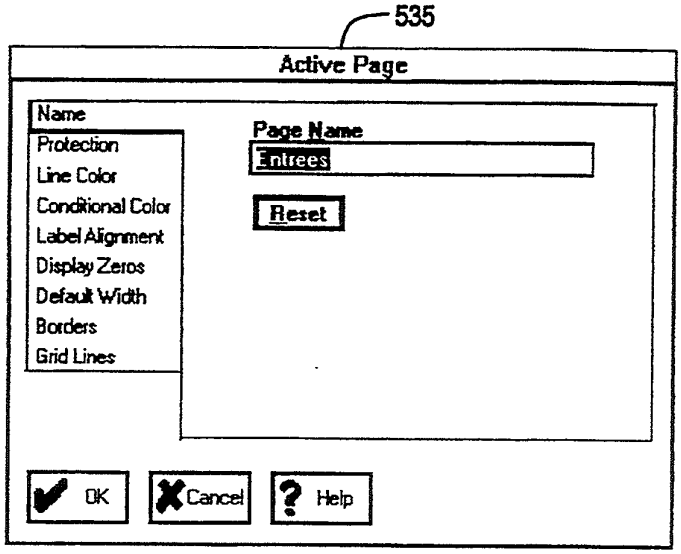


FIG. 5D

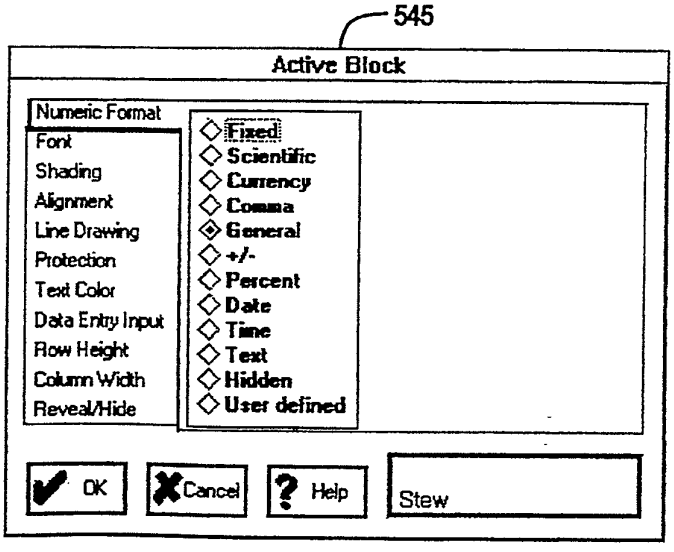


FIG. 5E

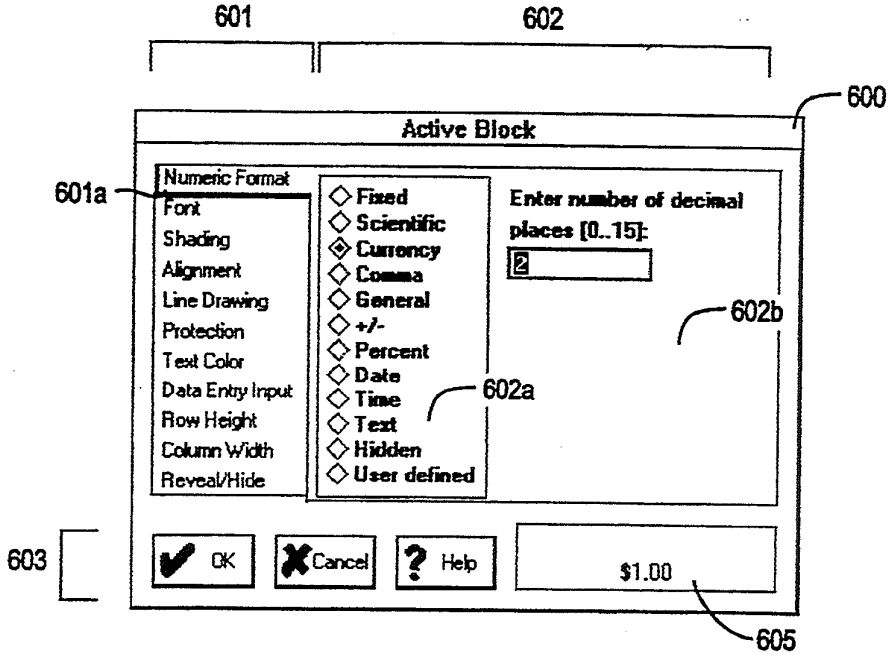


FIG. 6A

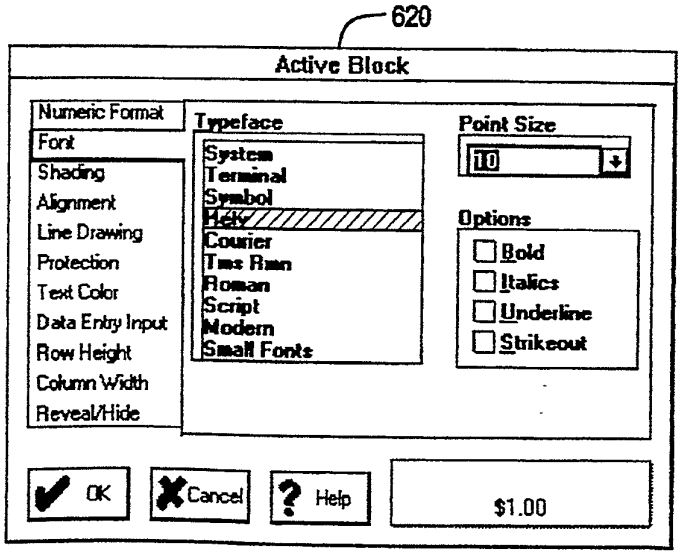


FIG. 6B

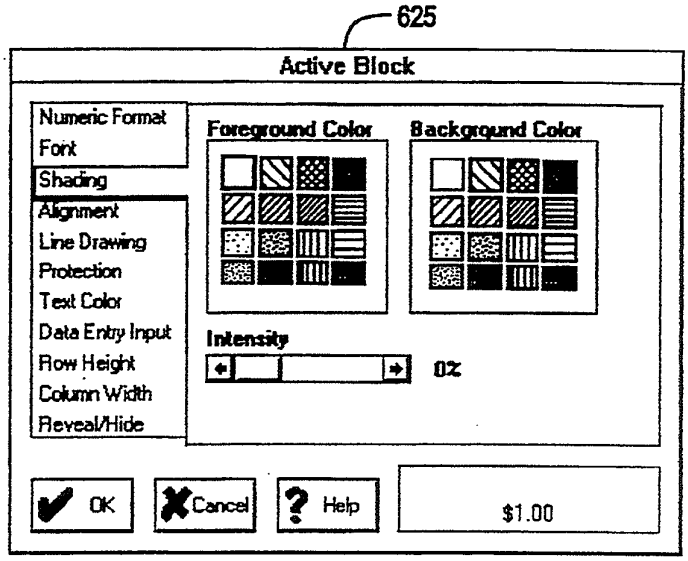


FIG. 6C

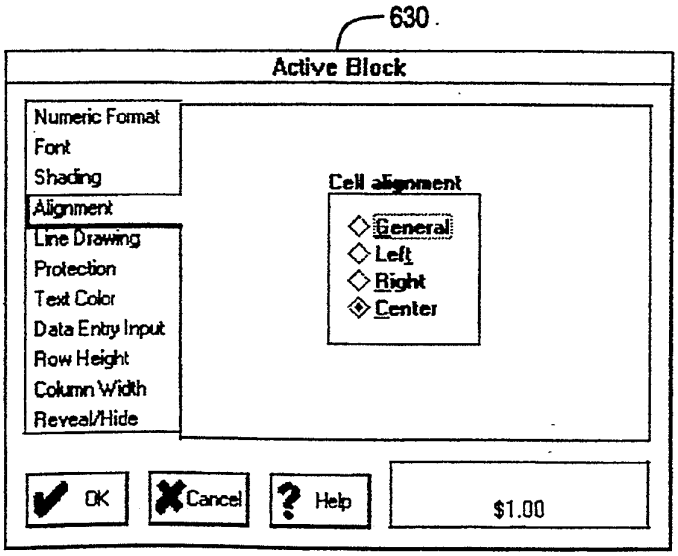


FIG. 6D

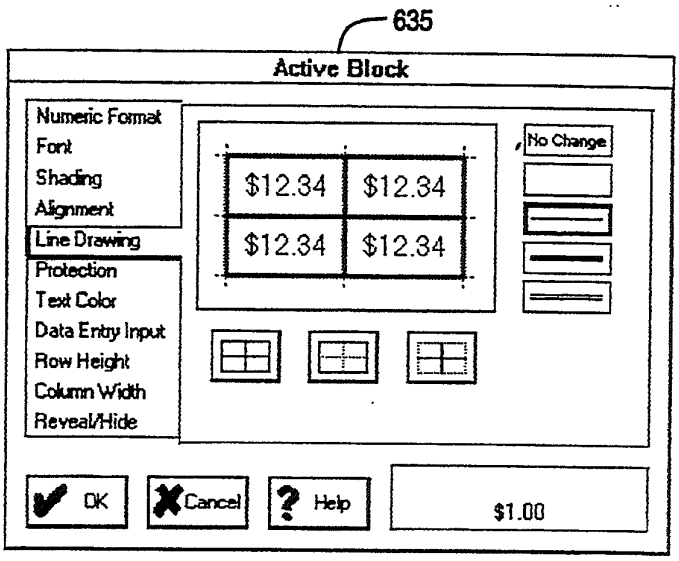


FIG. 6E

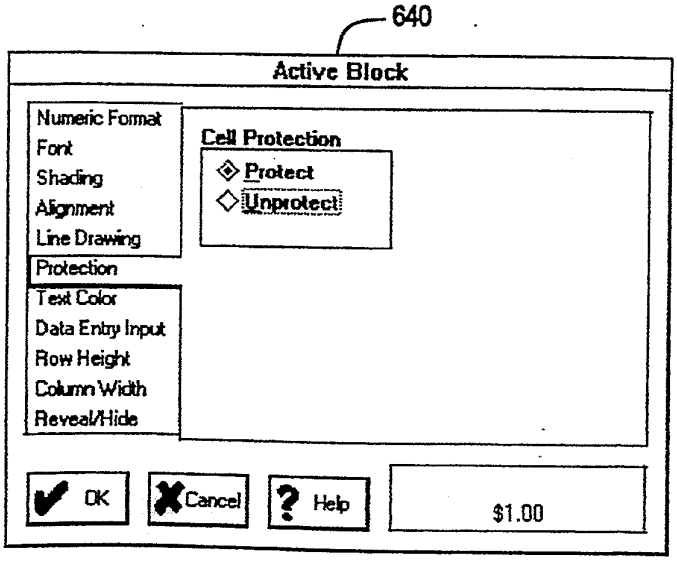


FIG. 6F

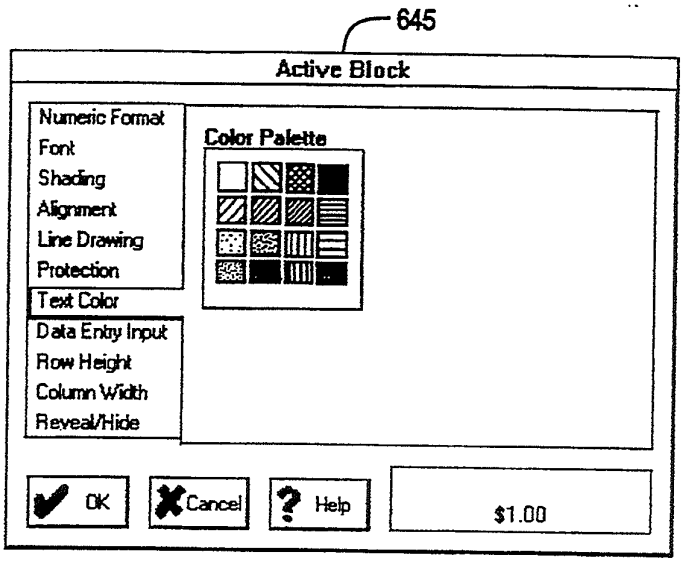


FIG. 6G

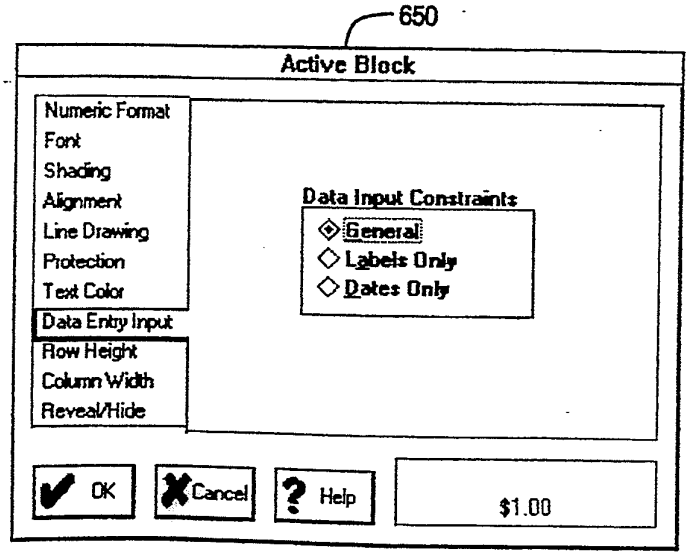


FIG. 6H

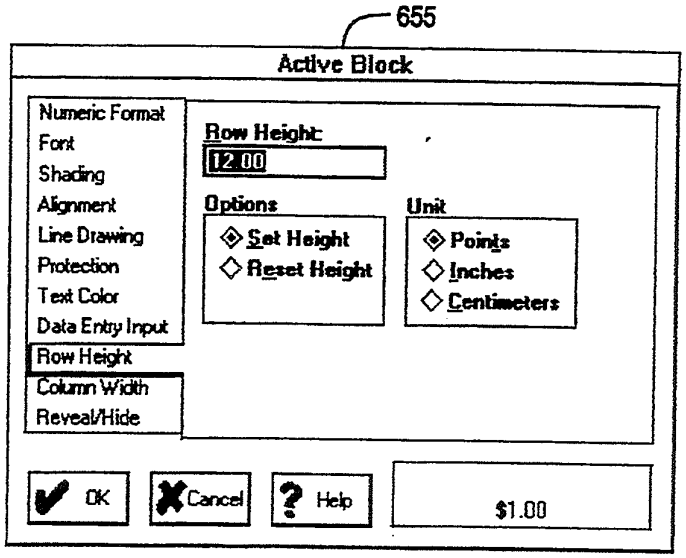


FIG. 6I

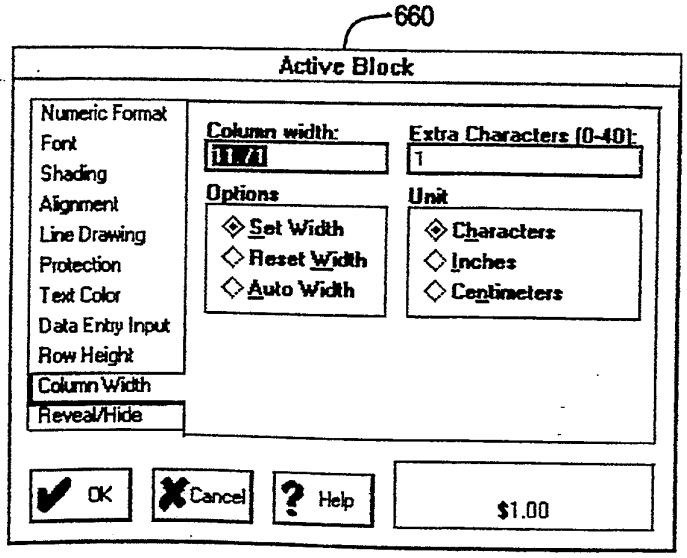


FIG. 6J

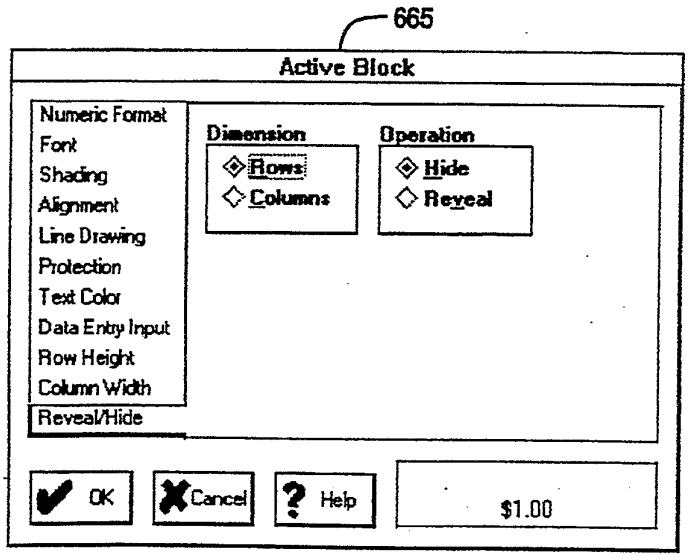


FIG. 6K

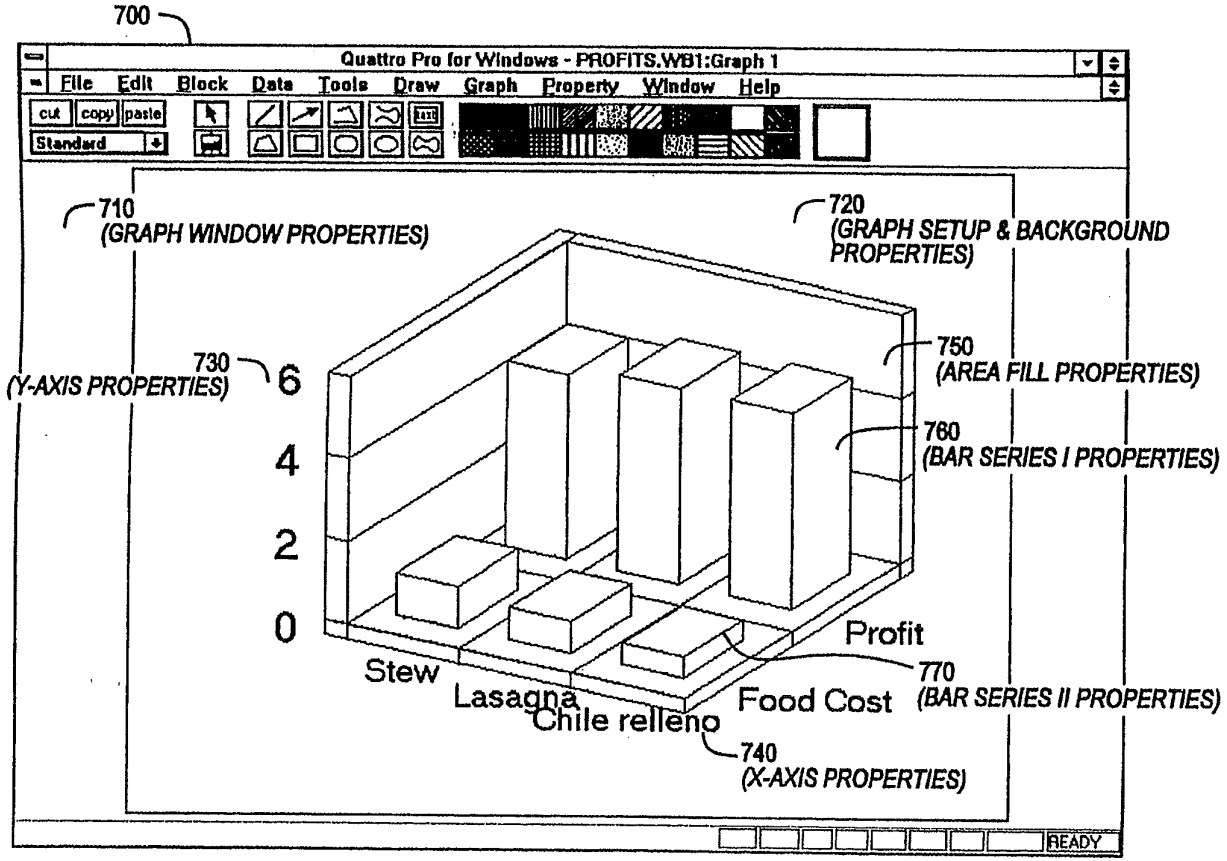

700
Quattro Pro for Windows - PROFITS.WB1:Graph 1
File Edit Block Data Tools Draw Graph Property Window Help
cut copy paste
Standard
710
(GRAPH WINDOW PROPERTIES)
720
(GRAPH SETUP & BACKGROUND PROPERTIES)
730
(Y-AXIS PROPERTIES)
6
4
2
0
750
(AREA FILL PROPERTIES)
760
(BAR SERIES I PROPERTIES)
Profit
770
(BAR SERIES II PROPERTIES)
Food Cost
Stew
Lasagna
Chile relleno
740
(X-AXIS PROPERTIES)
READY

FIG. 7A

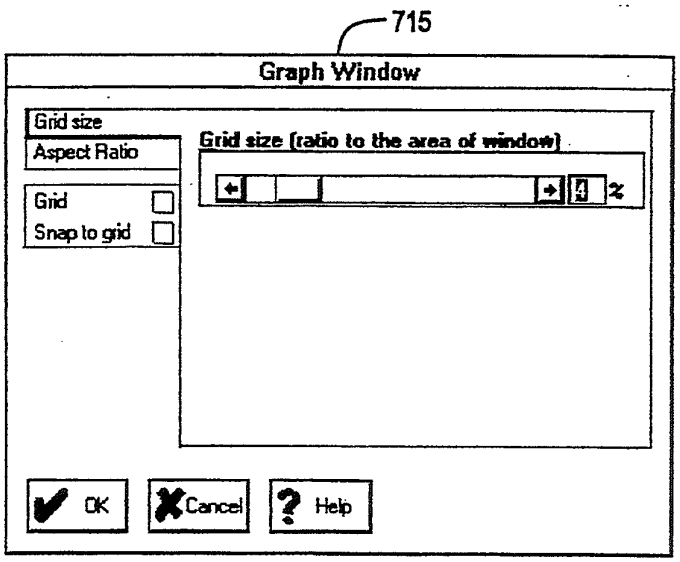


FIG. 7B

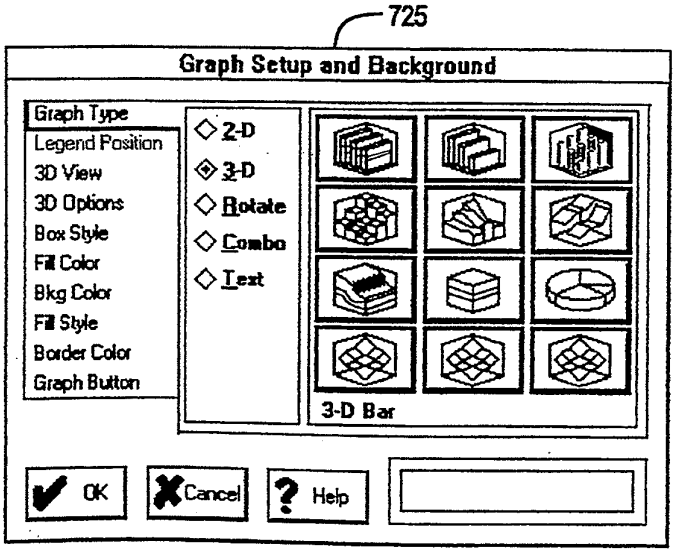


FIG. 7C

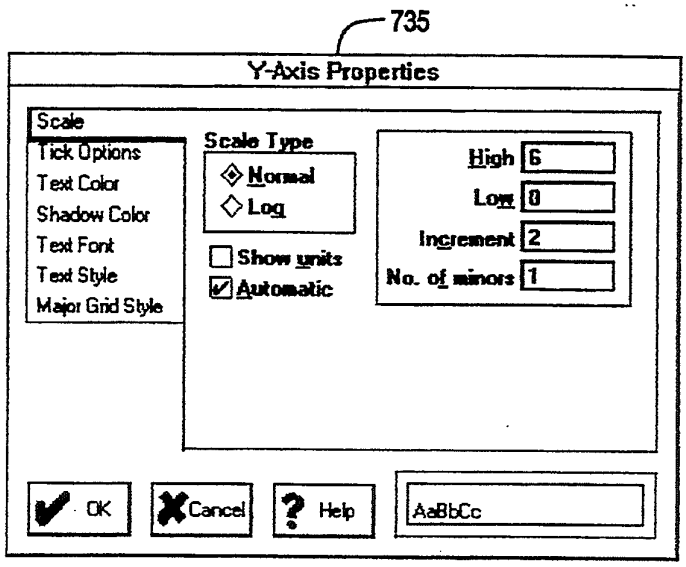


FIG. 7D

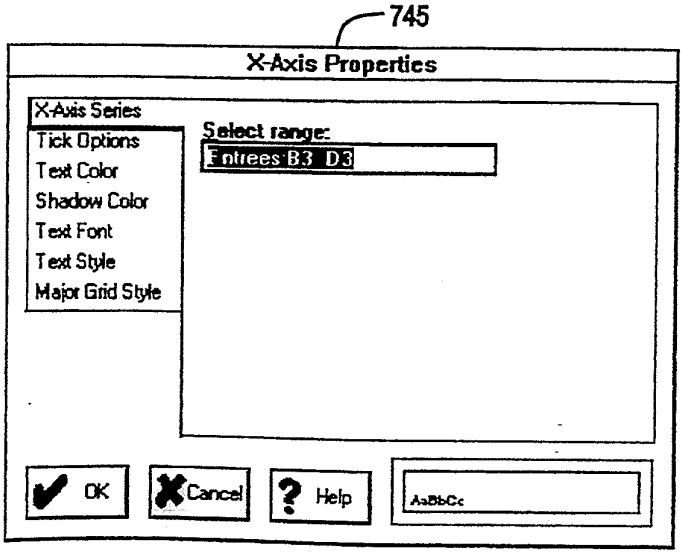


FIG. 7E

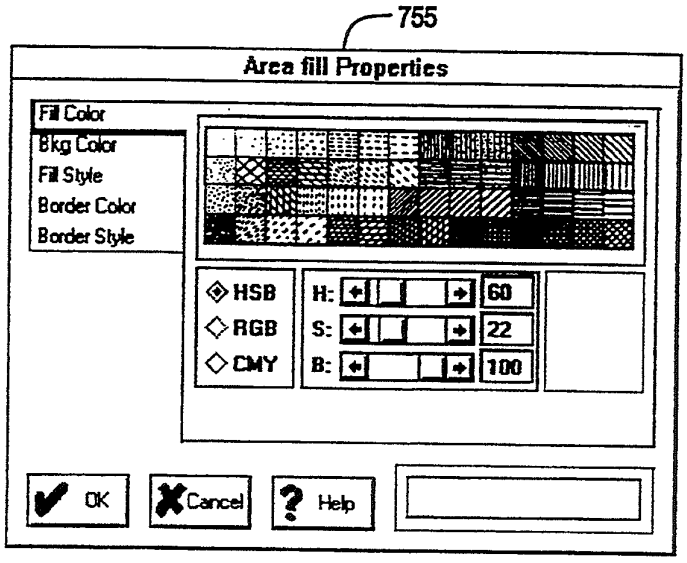


FIG. 7F

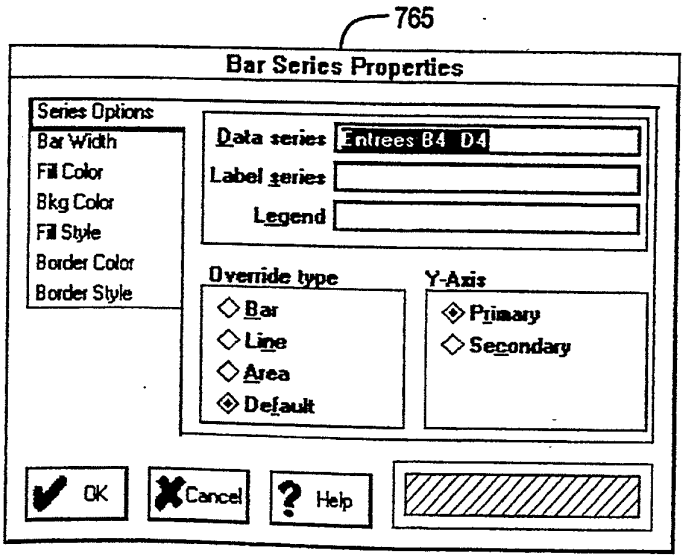


FIG. 7G

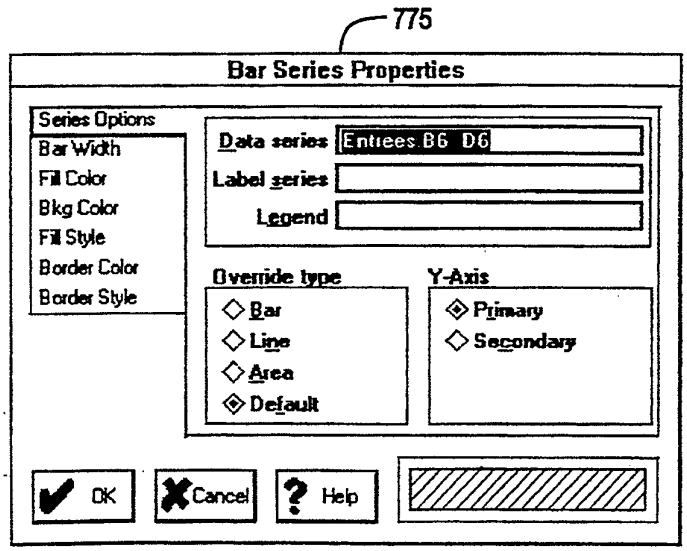


FIG. 7H

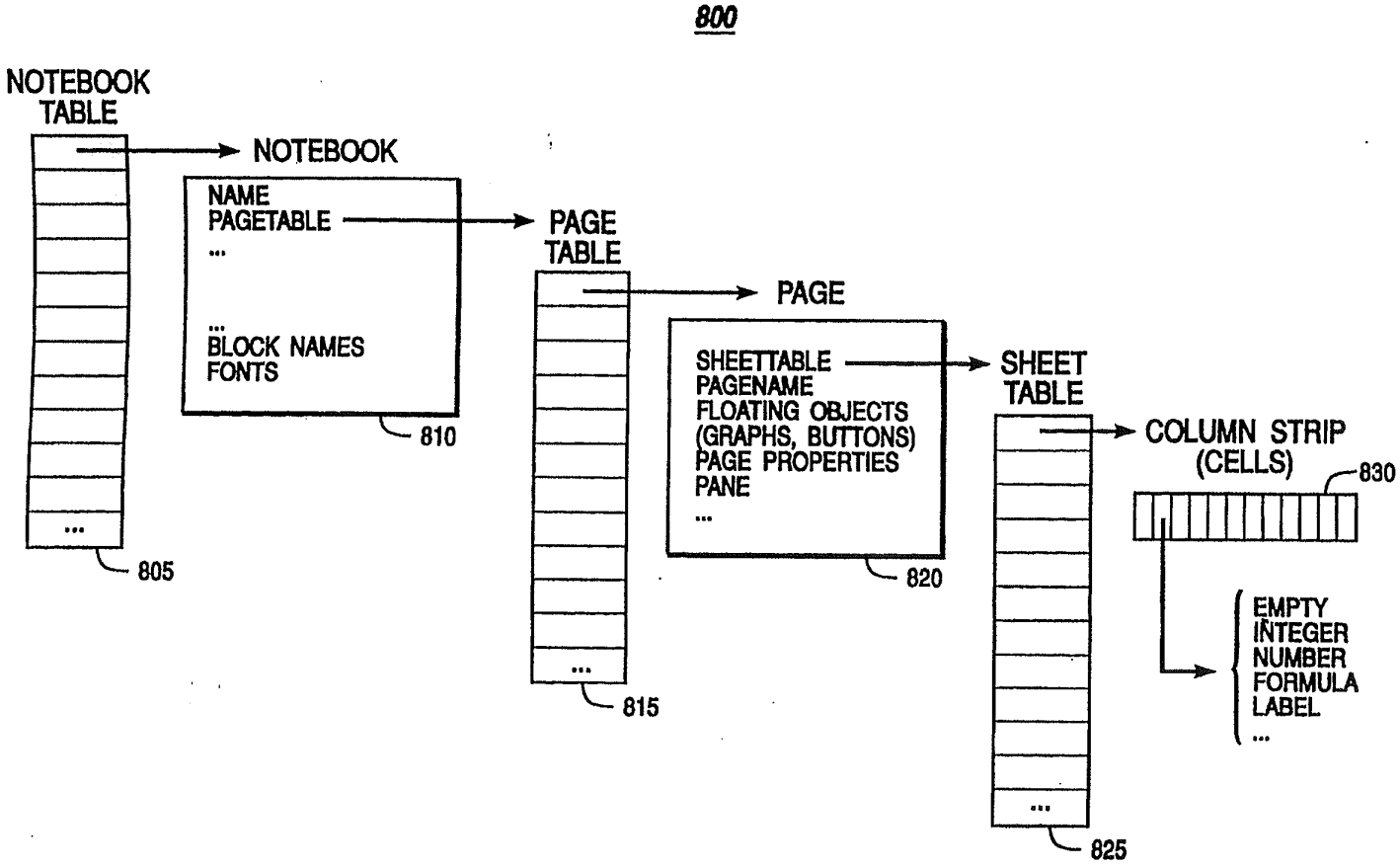


FIG. 8A

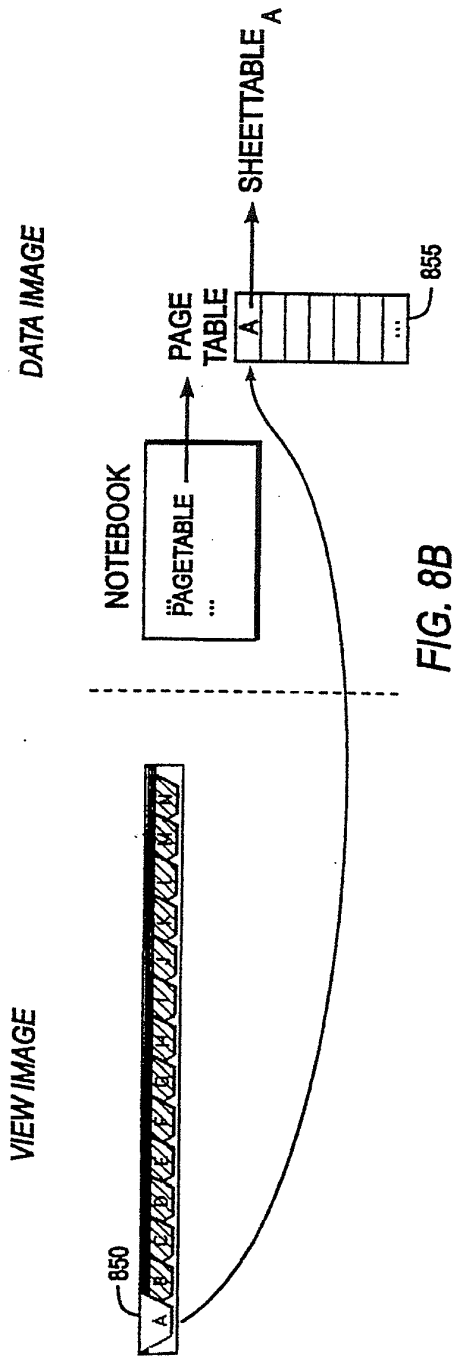


FIG. 8B

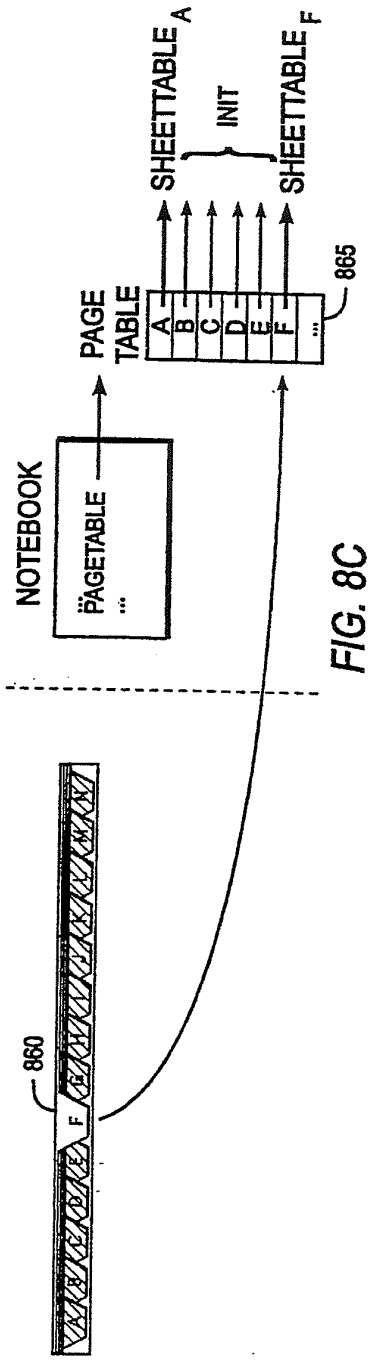


FIG. 8C

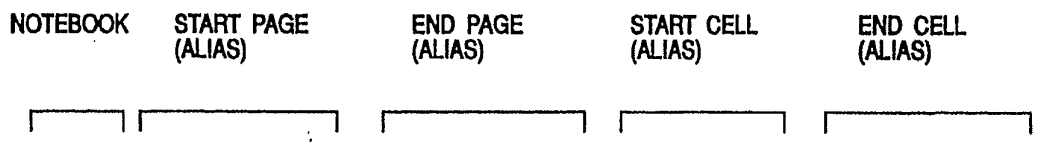


FIG. 8D

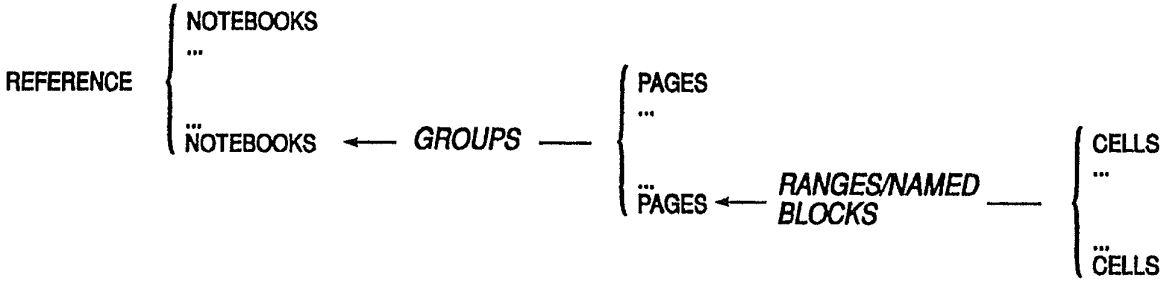


FIG. 8E

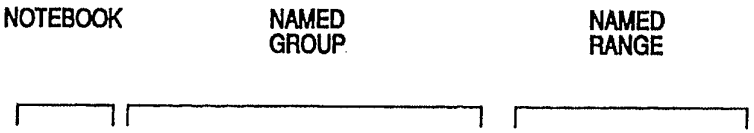


FIG. 8F

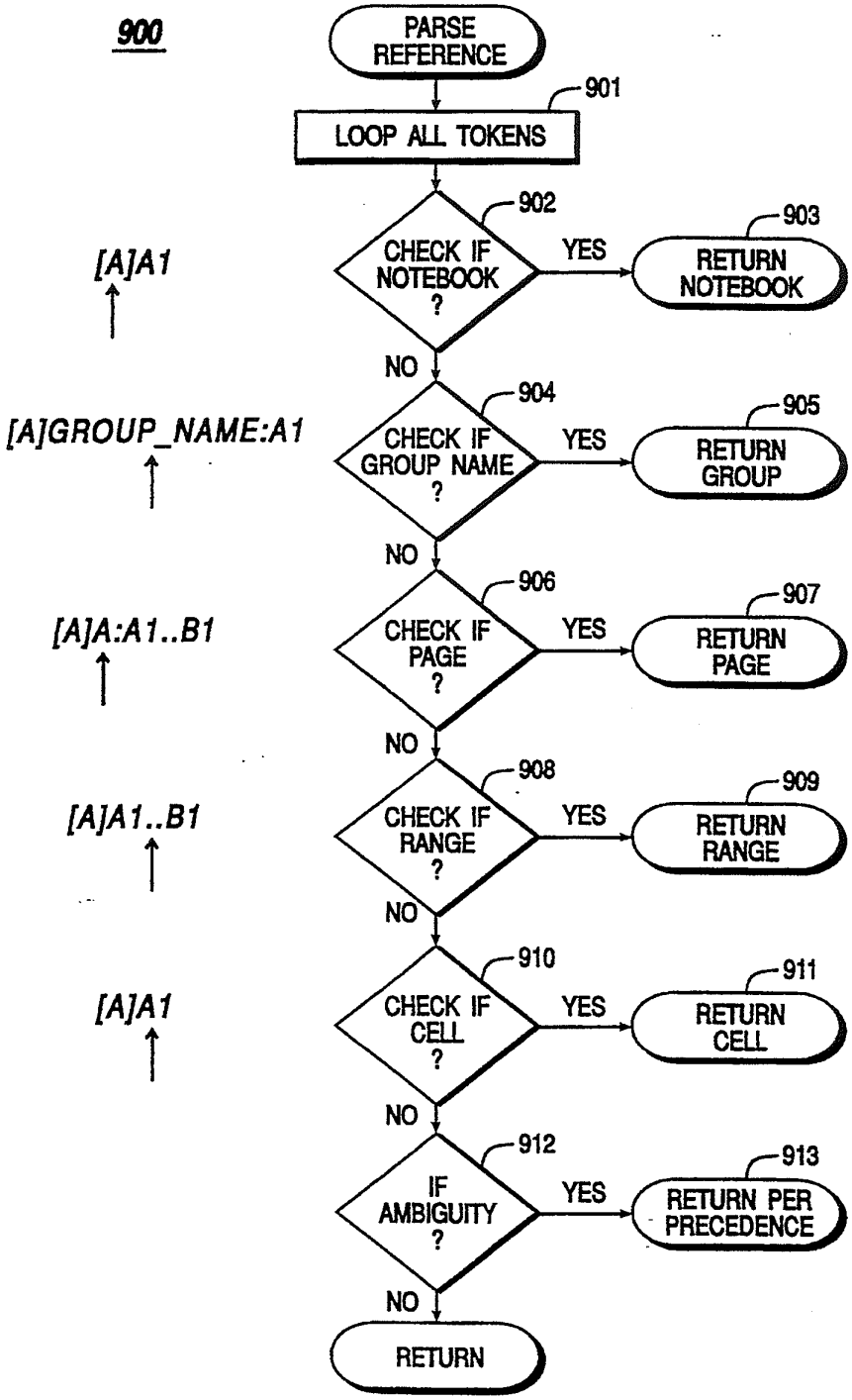


FIG. 9A

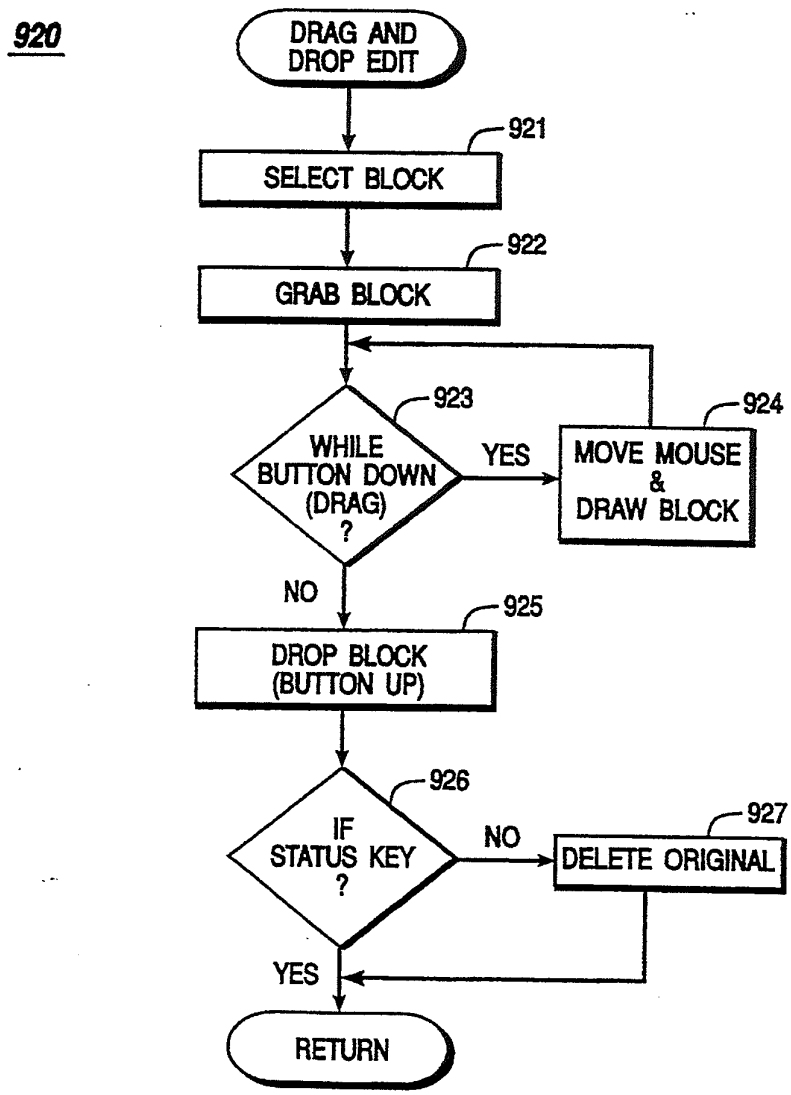


FIG. 9B

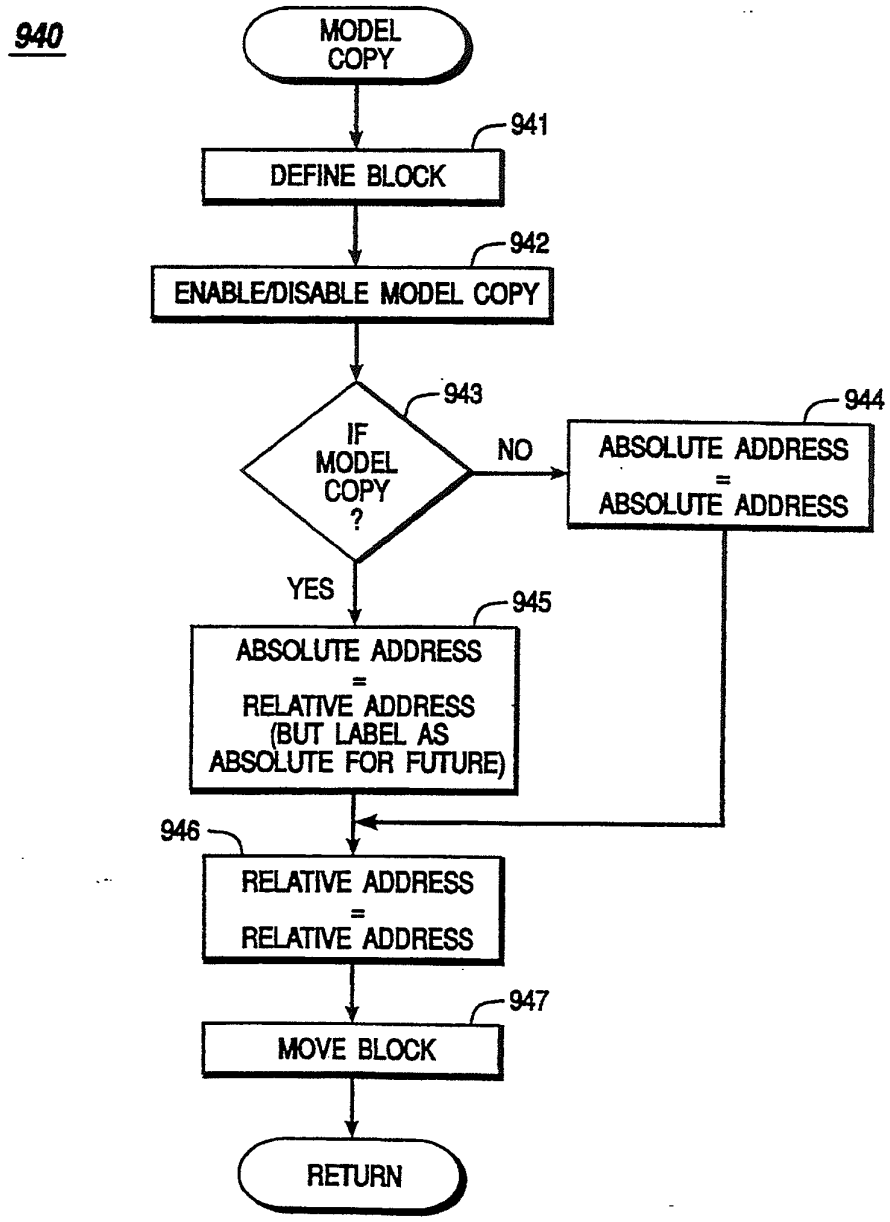


FIG. 9C

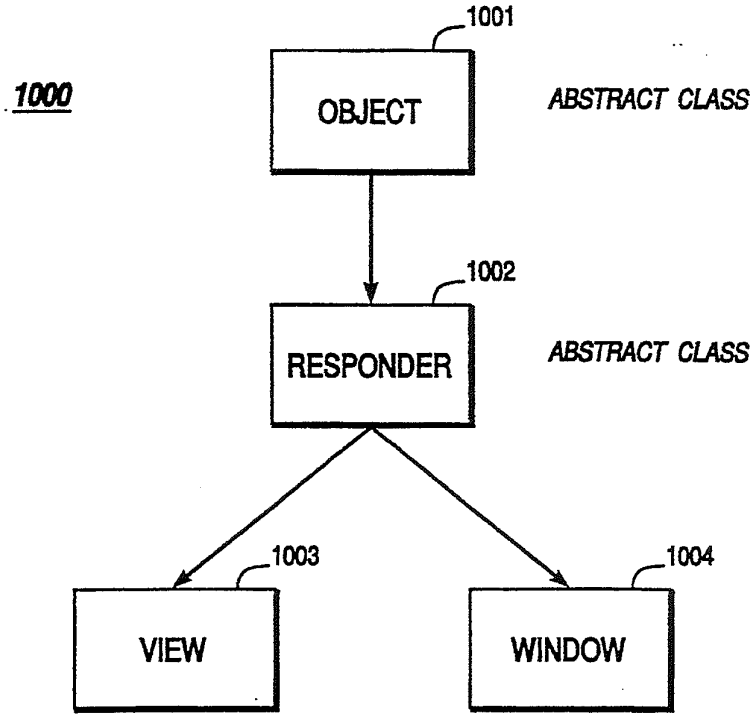


FIG. 10A

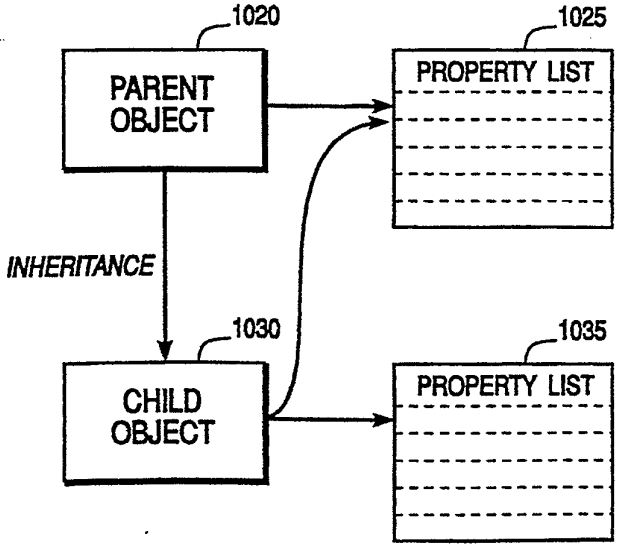


FIG. 10B

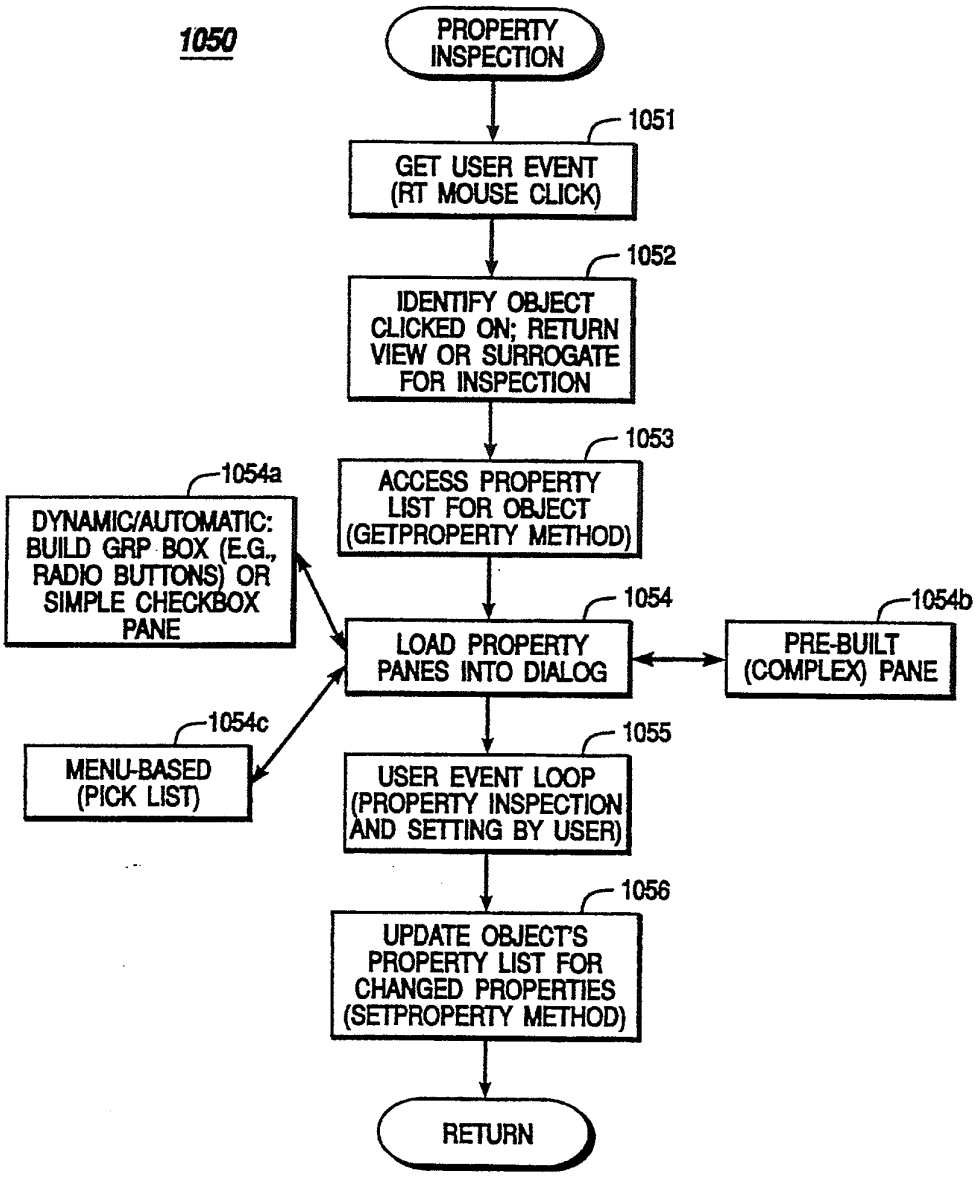


FIG. 10C

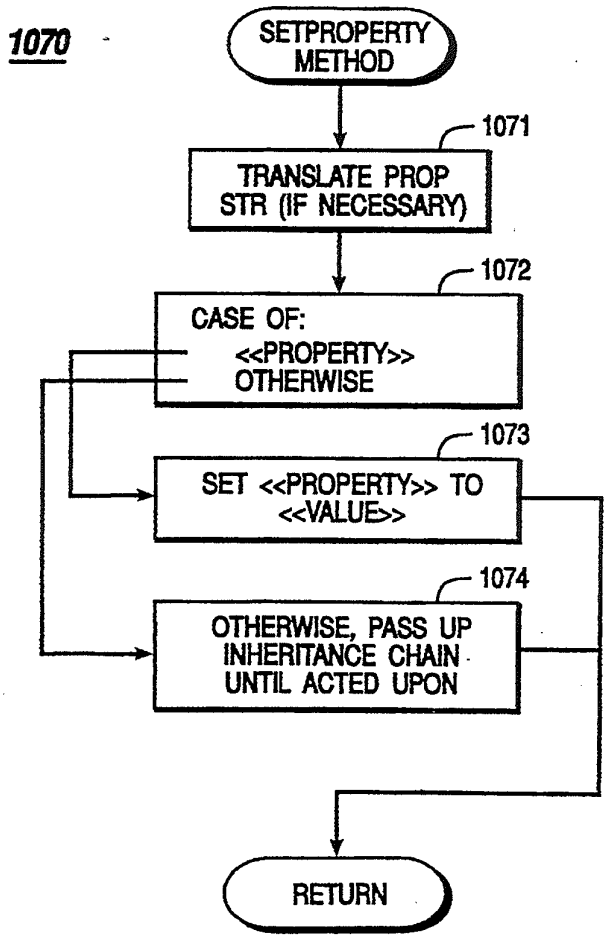


FIG. 10D

SYSTEM AND METHODS FOR IMPROVED SPREADSHEET INTERFACE WITH USER-FAMILIAR OBJECTS

COPYRIGHT NOTICE INTERFACE

MICROFICHE APPENDIX

A single-fiche microfiche Appendix A is includes with this application.

BACKGROUND OF THE INVENTION

The present invention relates generally to the field of information processing by digital computers and, more particularly, to the processing and presentation of information by program applications, particularly electronic spreadsheets.

Before computers, numerical analyses, particularly financial ones, were usually prepared on an accountant's columnar pad or spreadsheet, with pencil and calculator in hand. By organizing data into columns and rows, spreadsheets afford the rapid assimilation of information by a reader. The task of preparing a spreadsheet on paper, however, is not quite so fast. Instead, the process tends to be very slow, as each entry must be tediously calculated and entered into the spreadsheet. Since all calculations are the responsibility of the preparer, manually prepared spreadsheets are also prone to errors. Hence, preparation of spreadsheets by hand is slow, tedious, and unreliable.

With the advent of microcomputers, a solution was forthcoming in the form of "electronic spreadsheets." Better known simply as "spreadsheets," these software programs provide a computerized replacement for the traditional financial modeling tools: the accountant's columnar pad, pencil, and calculator. In some regards, spreadsheet programs are to those tools what word-processors are to typewriters. Spreadsheets offer dramatic improvements in ease of creating, editing, and using financial models.

A typical spreadsheet program configures the memory of a computer to resemble the column/row or grid format of an accountant's columnar pad, thus providing a visible calculator for a user. Because this "pad" exists dynamically in the computer's memory, however, it differs from paper pads in several important ways. Locations in the electronic spreadsheet, for example, must be communicated to the computer in a format which it can understand. A common scheme for accomplishing this is to assign a number to each row in a spreadsheet, and a letter to each column. To reference a location at column A and row 1 (i.e., the upper-lefthand corner), for example, the user types in "A1". In this manner, the spreadsheet defines an addressable storage location or "cell" at each intersection of a row with a column.

Data entry into an electronic spreadsheet occurs in much the same manner that information would be entered on an accountant's pad. After a screen cursor is positioned at a desired location, the user can enter alphanumeric information. Besides holding text and numeric information, however, spreadsheet cells can store

special instructions or "formulas" specifying calculations to be performed on the numbers stored in spreadsheet cells. In this fashion, cell references can serve as variables in an equation, thereby allowing precise mathematical relationships to be defined between cells. The structure and operation of a spreadsheet program, including advanced functions such as functions and macros, are documented in the technical, trade, and patent literature. For an overview, see e.g., Cobb, S., *Using Quattro Pro* 2, Borland-Osborne/McGraw-Hill, 1990; and LeBlond, G. and Cobb, D., *Using* 1-2-3, Que corp., 1985. The disclosures of each of the foregoing references are hereby incorporated by reference.

Electronic spreadsheets offer many advantages over their paper counterparts. For one, electronic spreadsheets are much larger (i.e., hold more information) than their paper counterparts; electronic spreadsheets having thousands or even millions of cells are not uncommon. Spreadsheet programs also allow users to perform "what if" scenarios. After a set of mathematical relationships has been entered into a worksheet, the spread of information can be recalculated using different sets of assumptions, with the results of each recalculation appearing almost instantaneously. Performing this operation manually, with paper and pencil, would require recalculating every relationship in the model with each change made.

While electronic spreadsheets offer significant productivity gains in the task of complex data modeling, none has been as intuitive to use as ordinary paper and pencil—objects already familiar to the user. Instead, the user must master many complex and arbitrary operations. To find the proper command for a task at hand, for example, the user must hunt through a complex menuing system, with the desired choice often buried under several menus. Even simple tasks can pose a significant challenge to the user. To change the punctuation format of a number in one prior art spreadsheet, for example, the user must traverse several nodes of a menu tree, carefully selecting among cryptic menu choices along the way. A mistake at any one of the nodes can lead to harsh consequences, including the loss of valuable data.

Finding this approach to be unworkable, many users memorize frequently-needed commands instead. To accomplish the foregoing task, for example, the user would memorize the command: /Worksheet Global Default Other International. As one can only memorize just so many arbitrary commands, however, the user typically masters only a very small subset of available commands and features. And without constant practice, these commands tend to be quickly forgotten. Moreover, many useful and needed commands are sufficiently hidden in layers of menus that they are never discovered by the user. All told, the non-intuitive interfaces of prior art spreadsheets have led to steep learning curves for users. Even after mastering a particular spreadsheet interface, the user typically only knows a fraction of available commands and features, most of which are easily forgotten.

Even with advances in computer and software technology, electronic spreadsheets have not necessarily become easier to use. Instead, technological advances have been largely employed to build more complex functions and 10 modeling features into spreadsheets, often with more complicated menu trees; or worse yet, a staggering array of icons which leave the user even

more bewildered. Thus, while prior art spreadsheets have continued to increase in functionality, they have also greatly increased in complexity for the user.

Three dimensionality is one such example. Three-dimensional spreadsheets allow the user to create a worksheet having cells arranged in a 3-D grid. In this manner, the user can manipulate multi-dimensional ranges, i.e., solid blocks of cells. This feature has distinct advantages. For example, the user can build a worksheet consisting of multiple two-dimensional spreads, define 3-D ranges that span these spreads, and copy a range of rows and columns into each of many 2-D spreads at once. This feature eases difficult choirs, such as consolidation of multiple spreads.

Despite its advantages, three-dimensionality, as presently implemented, is an advanced feature beyond the grasp of many spreadsheet users. This is not a necessary result of a three-dimensional model per se but, instead, has resulted from poor implementations of the model in prior art spreadsheet programs. One popular implementation of the model in the prior art, for example, requires the user to manipulate each additional spread of a three-dimensional spreadsheet as a separate window in a graphical windowing environment. This approach is far from intuitive, however. In particular, this approach requires the user to master actions which have no counterpart in everyday activities. While three-dimensional spreadsheets provide additional functionality, they serve to illustrate how non-intuitive implementations of new technology greatly add to the complexity of the user interface.

What is needed is application software which maintains a highly intuitive interface. Moreover, the application should implement advanced features, such as three dimensionality, by employing interface objects which are familiar to the user. In this manner, the user will not have to master an elaborate and/or awkward environment but, instead, may rely upon his or her own common fund of knowledge. The present invention fulfills this and other needs.

SUMMARY OF THE INVENTION

Application software, such as electronic spreadsheets, have found wide application in the processing and presentation of information. Despite their computational power, however, these programs require users to master complex interfaces, adopting abstract representation models which are beyond the grasp of ordinary users. As a result, much of the computational power of these applications goes un-utilized by end users.

The present invention, in contrast, provides system and methods having a highly intuitive interface for users. More particularly, the present invention provides interface objects which are familiar to the user. In an exemplary embodiment of the present invention, an electronic spreadsheet system includes a notebook interface having a plurality of notebook pages, each of which contains a spread of information cells, or other desired page type (e.g., Graphs page).

Methods are provided for rapidly accessing and processing information on the different pages, including, for example, displaying a plurality of page identifiers for selecting individual pages. Additional methods are provided for editing cells and blocks of cells. In this fashion, the spreadsheet notebook of the present invention provides a convenient means for organizing and presenting information. Moreover, the spreadsheet notebook; of the present invention readily accommodates complex data (e.g., consolidation across multiple spreads); yet, at the same time provides the user with a highly intuitive interface, one which employs interface objects which are familiar to the user.

The present invention also includes system and methods for conveniently inspecting and setting the properties of objects. A method for accessing an objects property, in accordance with the present invention, includes receiving a request from the user for inspection of an object; accessing properties for the object; and displaying the properties to the user. From the displayed properties, the user may alter the characteristics of the object, as desired.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1A is a block diagram of a computer system in which the present invention may be embodied.

FIG. 1B is a block diagram of a software system of the present invention, which includes operating system, application software, and user interface components.

FIG. 1C is bitmap screen shots illustrating the basic architecture and functionality of a graphical user interface in whip, the present invention may be embodied.

FIG. 2A is a screen bitmap illustrating a spreadsheet notebook of the present invention.

FIG. 2B is a bitmap of a tool bar component of the spreadsheet of the present invention.

FIG. 2C is a screen bitmap illustrating a notebook window from the spreadsheet notebook of the present invention.

FIGS. 2D–E are bitmaps illustrating page identifiers for rapidly accessing and manipulating individual pages of the notebook of the present invention.

FIGS. 3A–C illustrate navigation (i.e., access of desired information) in the spreadsheet notebook of the present invention.

FIGS. 4A–E are screen bitmaps illustrating the selection of information cells in the spreadsheet notebook of the present invention.

FIG. 4F is a screen bitmap illustrating the operation of grouping one or more desired pages in the spreadsheet notebook of the present invention.

FIG. 4G is a screen bitmap illustrating drag-and-drop editing in the spreadsheet notebook of the present invention.

FIGS. 4H–J are a series of screen bitmaps illustrating the model copy method of the present invention.

FIG. 4K is a set of bitmaps illustrating the operation of movingly and copying a notebook page.

FIGS. 4L–M are screen bitmaps illustrating the presentation of information in a notebook of the present invention.

FIG. 5A is a screen bitmap illustrating exemplary objects of the spreadsheet notebook of the present invention.

FIGS. 5B–E are bitmaps illustrating property inspectors of the present invention, for the objects of FIG. 5A.

FIGS. 6A–K are bitmaps illustrating different states (and accompanying panes) for the property inspector of FIG. 5E, each state depending on a particular property of the object being inspected.

FIG. 7A is a screen bitmap illustrating a graph window of the present invention with different graph objects available for property inspection.

FIGS. 7B–H are bitmaps illustrating exemplary property inspectors of the present invention for the graph objects of FIG. 7A.

FIG. 8A is a block diagram illustrating the structure and operation of a spreadsheet notebook of the present invention.

FIGS. 8B–C illustrate the correspondence between a page table data structure and pages which are displayed on the screen to the user.

FIGS. 8D–F illustrate the referencing of information in the spreadsheet notebook of the present invention.

FIG. 9A is a flowchart illustrating a method of the present invention for interpreting information references.

FIG. 9B is a flowchart illustrating a method of the present invention for drag and drop editing.

FIG. 9C is a flowchart illustrating a method of the present invention for model copying.

FIG. 10A is a block diagram illustrating a system class hierarchy of the present invention, which is employed for property inspection.

FIG. 10B is a block diagram illustrating the correspondence between a parent object and a child object, and their respective property lists.

FIG. 10C is a flowchart illustrating a method for inspecting and setting properties of objects, in accordance with the present invention.

FIG. 10D is a flowchart illustrating a set property method of the present invention (substep from the method of FIG. 10C).

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

System Hardware

As shown in FIG. 1A, the present invention may be embodied on a computer system such as the system **100**, which comprises a central processor **101**, a main memory **102**, an input/output controller **103**, a keyboard **104**, a pointing device **105** (e.g., mouse, track ball, pen device, or the like), a display device **106**, and a mass storage **107** (e.g., hard disk). Additional input/output devices, such as a printing device **108**, may be included in the system **100** as desired. As illustrated, the various components of the system **100** communicate through a system bus **110** or similar architecture. In a preferred embodiment, the computer system **100** includes an IBM-compatible personal computer, which is available from several vendors (including IBM of Armonk, N.Y.).

Illustrated in FIG. 1B, a computer software system **150** is provided for directing the operation of the computer system **100**. Software system **150**, which is stored in system memory **102** and on disk memory **107**, includes a kernel or operating system **151** and a shell or interface **153**. One or more application programs, such as application software **152**, may be "loaded" (i.e., transferred from storage **107** into memory **102**) for execution by the system **100**. The system **100** receives user commands and data through user interface **153**; these inputs may then be acted upon by the system **100** in accordance with instructions from operating module **151** and/or application module **152**. The interface **153**, which is preferably a graphical user interface (GUI), also serves to display results, whereupon the user may supply additional inputs or terminate the session. In a preferred embodiment, operating system **151** is MS-DOS, and interface **153** is Microsoft® Windows; both are available from Microsoft Corporation of Redmond, Wash. Application module **152**, on the other hand, includes a spreadsheet notebook of the present invention (as described in further detail hereinbelow).

Interface: User-familiar Objects

A. Introduction

The following description will focus on the presently preferred embodiments of the present invention, which are embodied in spreadsheet applications operative in the Microsoft Windows environment. The present invention, however, is not limited to any particular application or any particular environment. Instead, those skilled in the art will find that the system and methods of the present invention may be advantageously applied to a variety of system and application software, including database management systems, wordprocessors, and the like. Moreover, the present invention may be embodied on a variety of different platforms, including Macintosh, UNIX, NextStep, and the like. Therefore, the description of the exemplary embodiments which follows is for purposes of illustration and not limitation.

Referring now to FIG. 1C, the system **100** includes a windowing interface or workspace **160**. Window **160** is a rectangular, graphical user interface (GUI) for display on screen **106**; additional windowing elements may be displayed in various sizes and formats (e.g., tiled or cascaded), as desired. At the top of window **160** is a menu bar **170** with a plurality of user-command choices, each of which may invoke additional submenus and software tools for use with application objects. Window **160** includes a client area **180** for displaying and manipulating screen objects, such as graphic object **181** and text object **182**. In essence, the client area is a workspace or viewport for the user to interact with data objects which reside within the computer system **100**.

Windowing interface **160** includes a screen cursor or pointer **185** for selecting and otherwise invoking screen objects of interest. In response to user movement signals from the pointing device **105**, the cursor **185** floats (i.e., freely moves) across the screen **106** to a desired screen location. During or after cursor movement, the user may generate user-event signals (e.g., mouse button "clicks" and "drags") for selecting and manipulating objects, as is known in the art. For example, Window **160** may be closed, resized, or scrolled by "clicking" (selecting) screen components **172**, **174/5**, and **177/8**, respectively.

In a preferred embodiment, screen cursor **185** is controlled with a mouse device. Single-button, double-button, or triple-button mouse devices are available from a variety of vendors, including Apple Computers of Cupertino, Calif., Microsoft Corporation of Redmond, Wash., and Logitech Corporation of Fremont, Calif., respectively. More preferably, screen cursor control device **105** is a two-button mouse device, including both right and left "mouse buttons." Programming techniques and operations for mouse devices are well documented in the programming and hardware literature; see e.g., *Microsoft Mouse Programmer's Reference*, Microsoft Press, 1989, the disclosure of which is hereby incorporated by reference.

B. Spreadsheet Notebooks

In contrast to conventional applications, even those operative in a windowing environment, the present invention includes user-familiar objects, i.e., paradigms of real-world objects which the user already knows how to use. In an exemplary embodiment, system **100** includes a spreadsheet notebook interface, with such user-familiar objects as pages and tabs. In this manner, complexities of the system are hidden under ordinary, everyday object metaphors. By drawing upon skills which the user has already mastered, the present inven-

tion provides a highly intuitive interface—one in which advanced features (e.g., three-dimensionality) are easily learned.

1. General interface

Shown in FIG. 2A, a spreadsheet notebook interface of the present invention will now be described. The spreadsheet notebook or workbook of the present invention includes a notebook workspace **200** for receiving, processing, and presenting information, including alphanumeric as well as graphic information. Notebook workspace **200** includes a menu bar **210**, a tool bar **220**, a current cell indicator **230**, an input line **231**, a status line **240**, and a notebook window **250**. The menu bar **210** displays and invokes, in response to user inputs, a main level of user commands. Menu **210** also invokes additional pulldown menus, as is known in windowing applications. Input line **231** accepts user commands and information for the entry and editing of cell contents, which may include data, formulas, macros, and the like. Indicator **230** displays an address for the current cursor (i.e., active cell) position. At the status line **240**, system **100** displays information about the current state of the workbook; for example, a "READY" indicator means that the system is ready for the user to select another task to be performed.

The tool bar **220**, shown in further detail in FIG. **2B**, comprises a row or palette of tools which provide a quick way for the user to choose commonly-used menu commands or properties. In an exemplary embodiment, tool bar **220** includes cut, copy, and paste buttons **221**, a power button tool **222**, a graph tool **223**, alignment buttons **224**, a style list **225**, font buttons **226**, insert-/delete buttons **227**, a fit button **228**, and action buttons **229**. Buttons **221** cut, copy and paste data and objects to and from Windows clipboard. The same actions are also available as corresponding commands in the Edit menu (available from menu bar **210**). Tool **220** creates "powerbuttons" which allow a user to run spreadsheet macros; in a specific embodiment, powerbuttons appear as floating objects in a layer above spreadsheet cells. In a similar fashion, the graph tool **223** creates floating graphs that appear above spreadsheet cells.

Other tools are available for formatting cells. Alignment buttons **224** place cell entries flush left, centered, or flush right, as desired. The style list **225** specifies the style for the active block and may be selected from a plurality of pre-defined styles (e.g., normal, currency, fixed, percent, and the like). The font buttons **226** effect font changes, including toggling bold and italic fonts, as well as increasing and decreasing font (point) size. The insert and delete buttons **227** permit the user to insert or delete blocks, rows, columns, and pages (described in further detail hereinbelow), as desired. The fit button **28** allows the user to quickly tailor a column's width to match its widest cell entry. Action buttons **229** provide automated spreadsheet operations, including sorting and summing operations. For example, a Sort button, when invoked, performs a sort on data in a currently active block. A sum button, on the other hand, totals the values in any block of cells by generating an @SUM function, just outside a selected range of blocks.

2. Notebook and pages

Notebook **250**, shown in further detail in FIG. **2C**, provides an interface for entering and displaying information of interest. The notebook **250** includes a plurality of spreadsheet pages, such as page **251** (Page A). Notebook **250** also includes a "Graphs page" for readily accessing all the graphs created for the notebook. Each page may include conventional windowing features and operations, such as moving, resizing, and deleting. In a preferred embodiment, the notebook **250** includes **256** spreadsheet pages and one Graphs page, all of which are saved as a single disk file on the mass storage **107**. In accordance with the present invention, workspace **200** may display one or more notebooks, each sized and positioned (e.g., tiled, overlaping, and the like) according to user-specified constraints. When a single notebook is used, notebook **250** will typically occupy a majority of workspace **200**.

Each spreadsheet page of a notebook includes a 2-D spread. For example, spreadsheet page **251** includes a grid in row and column format, such as row **252** (row 3) and column **255** (col. F). At each row/column intersection, a box or cell (e.g., cell C4) is provided for entering, processing, and displaying information in a conventional manner. Each cell is addressable, with a selector **253** being provided for indicating a currently active one (i.e., the cell that is currently selected).

As shown in FIGS. 2C–E, individual notebook pages are identified by page identifiers **260**, preferably located along one edge of the notebook **250**. In a preferred embodiment, each page identifier is in the form of a tab member (e.g., members **261***a*, **262***a*, **263***a*) situated along a bottom edge of the notebook. Each tab member may include representative indicia, such as textual or graphic labels, including user-selected titles representing the contents of a corresponding page. In FIG. 2D, the tab members **260** are set to their respective default names. For example, the first three tab members (members **261***a*, **262***a*, **263***a*) are respectively set to A, B, and C. In a preferred embodiment, however, tab members are typically given descriptive names provided by the user. As shown in FIG. 2E, for example, the first three tab members have now been set to "Contents" (tab member **261***b*), "Summary" (tab member **262**b), and "Jan" (tab member **263***b*). In a similar manner, the remaining tabs are set to subsequent months of the year. In this manner, the user associates the page identifiers with familiar tabs from an ordinary paper notebook. Thus, the user already knows how to select a page or spread of interest: simply select the tab corresponding to the page (as one would do when selecting a page from a paper notebook).

In addition to aiding in the selection of an appropriate page of information, the user-customizable page identifiers aid in the entry of spreadsheet formulas. For example, when entering a formula referring to cells on another page, the user may simply use the descriptive page name in the formula itself (as described hereinbelow), thus making it easier for the user to understand the relationship of the cell(s) or information being referenced.

3. Navigation in a notebook

Referring now to FIGS. 3A–C, movement (i.e., location of desired information cells) within a spreadsheet notebook of the present invention is illustrated. To move to different pages in the notebook, the user simply selects the corresponding tab from tabs **260**. To move to Page B, for example, the user selects (e.g., with keyboard **104** or pointing device **105**) tab **262***a*; similarly, Page C is reached by selecting tab **263***a*. Continuing the example, the user may return to Page A by selecting tab **261***a*. Thus instead of finding information by scrolling different parts of a large spreadsheet, or by invoking multiple windows of a conventional three-dimensional spreadsheet, the present invention allows the user to

simply and conveniently "flip through" several pages of the notebook to rapidly locate information of interest.

Notebook **250** also includes supplemental tools for navigation between pages, including a tab scroller **271** and a fast-forward button **272**. The tab scroller (shown in two different states: **271***a* and **271***c*) permits access to identifiers for pages which are not currently visible on the screen device **106**. If a desired identifier or tab is not currently in view, the user simply activates the tab scroller **271** to reveal additional tabs. The fast-forward button **272**, on the other hand, provides immediate access to the last pages of the notebook, including the Graphs page. As shown in FIG. 3A and B, after invoking the fast-forward button **272***a*, the page identifiers for the last pages (e.g., tabs **261***c*, **262***c*, **263***c*, **265**) are accessible. To switch back to the previously active spreadsheet page, the user may select or click the fast-forward button **272***c* again. For navigation within a spreadsheet page, horizontal and vertical scrollbars **278**, **259** are provided; general features and operations of basic scroller or sliders are described in Windows user and programming literature, including Microsoft's *Windows Software Development Kit.*

4. Selections and aggregate operations within a notebook

The selection of desired information cells in the notebook of the present invention is now described. For selecting a plurality of information cells, both 2-D blocks (e.g., block **254** of FIG. 2C) and 3-D blocks of cells may be defined. A 2-D block is a rectangular group of one or more cells and is identified by block coordinates, such as the cell addresses of its upper-left and bottom-right corners. Similarly, a 3-D block represents a solid block (e.g., cube) of cells.

A 2-D block is specified by selecting, with mouse **105** or keyboard **104**, opposing corners. In FIG. 2C, for example, the block **254** is defined by corner cells C5 and F14. Additional selection examples are illustrated in FIGS. 4A–E. [or example, column B (col. **411**) is selected by clicking column heading **410**; similarly, row 3 (row **421**) is chosen by clicking row heading **420**. Selection may be additive (i.e., additional selections are appended to the current ones), as shown by selection of a row **420** and a column **410** in FIG. 4C. To facilitate the selection of all cell members (e.g., block **431**), a select-all button **430** is also provided. In addition to these "contiguous" blocks, non-contiguous block selection (e.g., selection of blocks **441**, **442**) is provided by use of a status key (e.g., CTRL-, ALT-, or SHIFT-) plus a mouse event (e.g., click and drag operations).

Selection of 3-D cell blocks, i.e., cell ranges spanning more than one page, occurs in a similar fashion. To extend the block **254** (of FIG. 2C) into a 3-D block, the user specifies an additional or third dimension by selecting an appropriate page identifier. If the user selects the D tab while the block **254** is selected (e.g., with a SHIFT or other status key), the 2-D block is extended into a 3-D block spanning from Pages A to D. Additional 3-D operations may be performed by utilizing a method of the present invention for grouping pages, which will now be described.

Pages may be selected or grouped together, thereby providing a means for changing multiple pages simultaneously. In much the same manner as cells from a spread are grouped into 2-D blocks, a range of pages are grouped by specifying beginning and ending members. As shown in FIG. 4F, a range from Page A to Page K may be achieved by selecting tabs A (**261**) and K (**267**) from identifiers **260**, for example, while depressing a key (e.g., status key). A grouping indicator **268** is displayed for indicating members of a group; groupings may also be annotated with user-specified labels. Once grouped, a page of the group may have its operations (e.g., selection, data entry, and the like) percolate to the other members of the group, as desired. A non-contiguous selection of pages may also be selected (even across different pages); for example, a selection of Pages A and D, but not B and C, may be achieved by selecting tabs A and D while depressing a second key (e.g., CTRL-key). Furthermore, groups may overlap (i.e., a page can be in more than one group), as desired. By selectively activating a group mode (e.g., by toggling group button **273**), groupings may be temporarily turned off, in which case events are not percolated to other members of the group.

With group mode active, an activity in a page which is a member of a group can also affect similarly situated cells of the other pages of the group. For example, information entered in a cell on one page of the group can also propagate to the same (relative) cell in other pages of the group; data entry may be "drilled" (propagated) to other group pages by concluding data entry, for example, with a "Ctrl-Enter" keystroke (instead of a "Enter" command). For instance, an entry into cell C4 of Page A may also conveniently and automatically appear in cell C4 of pages B, C, D, E, F, G, H, I, J, and K, for an A–K grouping. Similarly, block or aggregate operations may propagate across member pages. For example, a block operation (e.g., Block Fill) for cells A1 to C4 in Page A in an A–D grouping would also operate (in this case, fill information) in the same cells (i.e., from A1 to C4) for Page B to Page D.

Employing the user-specified page identifiers of the present invention, a simple nomenclature is available for specifying these solid blocks of information. In a preferred embodiment, a solid block is specified by:

<<First Page>> . . . <<Last Page>>:
<<First Cell>> . . . <<Last Cell>>

For example, a solid block may be defined as A . . . D:A1 . . . C4, in which case the block spans from cells A1 to C4, and spans across Pages A–D. By permitting alias names (i.e., user-supplied alternative labels), the present invention allows the block to be specified as 1989 Sales . . . 1992 Sales: A1 . . . C4; or even 1989 Sales . . . 1992,Sales: First Quarter . . . Third Quarter. Additionally, the present invention readily accommodates notebook information as well, for example, [TAX]1989 Sales . . . 1992 Sales: First Quarter . . . Third Quarter, thus permitting information to be linked across various notebooks. Wildcard characters (e.g., * and ?) may also be employed for cell, page, and notebook identifiers, as desired. Thus, the spreadsheet notebook of the present invention provides a 3-D interface which readily accommodates real-world information in a format the user understands (instead of forcing the user to adapt his or her information to fit an arbitrary spreadsheet model).

5. Advanced Editing

Whether 2-D or 3-D in nature, blocks of cells may be easily copied and cut (i.e., moved) using drag-and-drop editing techniques of the present: invention. As shown in FIG. 4G for a 2-D block, for example, a method for copying a block of cells includes (1) selecting a source block by dragging a range of cells (e.g., mouse button-down events coupled with mouse movement across the

range; close selection with a button-up event), (2) dragging the block (e.g., click within block, followed by repeated mouse button-down events), and (3) dropping the block (e.g., mouse button-up event at desired target location). In a similar fashion, 3-D blocks may be dragged and dropped.

In typical cut and copy operations, relative and absolute cell addressing is employed, as is known in the art (see e.g., *Using* 1-2-3). According to the present invention, however, a "model copy" technique for copying blocks is also provided. Model copying, illustrated in FIGS. 4H–J, is useful when the user is copying a block that contains absolute references to cells within the copied block. In FIG. 4H, a small spread model **496** is shown which contains a formula to figure the monthly payment for a 30-year loan at different interest rates; a reference to the loan amount was entered as absolute so that when the formula is copied, it continues to refer to cell B1. The user may want to calculate (at the same time) monthly payments for different loan amounts and, thus, might copy the model, with the loan amount changed (shown in FIG. 4I as spread **497**). However, in this approach the absolute reference still refers to row 1; to correct this, the user would typically manually edit each formula to refer to B6 instead of B1.

With model copying of the present invention enabled (e.g., by default or through a user dialog), however, the problem is solved. In particular, absolute references adjust to the new location of the referenced cell, as shown by spread **498** in FIG. 4J; however, absolute references remain absolute to make future copies absolute. For instance, should the user make more copies of the formula, the reference to cell B6 is still absolute. In this manner, model copying of the present invention saves the user time-consuming and error-prone editing of formulas.

As shown in FIG. 4K, notebook pages may be copied or moved using the drag-and-drop editing techniques of the present invention (a dialog box is also provided as an alternative). As shown, for example, the "Salads" page is moved by selecting its tab identifier **264***a* from the identifiers **260**; in a preferred method, the identifier is selected by dragging it downward (mouse button-down plus downward movement) with the mouse cursor. Next, the identifier is dragged to a desired new position (mouse button-down plus left and right movement). Once positioned at a desired location (as indicated by in-transit tab identifier **264***b*), the page is then "dropped" (mouse button-up) into position. Similarly, a "copy" operation may be effected by coupling the above operation with a status key (e.g., CTRL-); in a preferred method of copying, only the page information (not its tab identifier) is copied to a target location.

Additional editing operations may be easily accomplish using the page identifiers of the present invention. For example, an additional page may be inserted by selecting a desired tab and entering "INS" (keyboard or mouse). Similarly, a notebook page may be deleted by coupling the selection of a corresponding page tab to a delete command.

6. Advantages

In contrast to prior art spreadsheet implementations, use of the spreadsheet notebook of the present invention is easily ascertained by the user. The notebook interface of the present invention provides a convenient means for organizing many spreadsheets together into one file. This permits the user to load (into memory **102**) all related information with a single command, without having to remember a variety of different file names. Moreover, the notebook interface **250** encourages a user to spread data out among multiple pages, thus better organizing one's data. In FIG. 4L, for example, spreadsheet page **300** illustrates the conventional method for grouping unrelated information onto a single spreadsheet. As shown in column B, the incorporation of unrelated information into a single spread leads to unnecessary whitespace padding of information. In FIG. 4M, in contrast, related information is grouped on separate pages **351**, **352** (e.g., with their own columns, Col. B_1 and Col. B_2) of notebook **350**. Not only does this eliminate grouping of disparate information, with its disadvantages (e.g., padding data entries), but it also encourages better organization of information.

Inspecting and Setting the Properties of Objects

A. Disadvantages of Prior Techniques

Standard windowing interfaces depend heavily on a clunky system of pull-down menus. While pull-down menus are an improvement over command-line (e.g., MS-DOS) interfaces, they are non-metaphoric or non-analogous to ordinary objects, i.e., ones familiar to the user. Thus, prior art windowing GUIs are no more intuitive or useful to the user than other menuing systems.

Perhaps the biggest weakness of pull-down menus is the requirement that the user must know beforehand which menu contains the item or function of interest. If the user does not know which pull-down menu, submenu, or dialog box contains the item he or she is seeking, the user will spend an inordinate amount of time checking the contents of each in an effort to hunt down the needed item. And often the search is in vain. Moreover, since functions for a given object (e.g., text object) are often scattered over several disparate menus, the user is discouraged from interacting and experimenting with the object.

One approach to overcoming this problem has been the implementation of "smart icons." This interface technique employs screen buttons painted with icons which are supposed to tell the user what the buttons do. This approach, however, often makes the interface problem even worse because the meaning of the icons are not easily grasped. Thus, the approach is often no more helpful than hiding the desired function deep inside a menuing system. Worse, some button images are so small or so numerous that it is hard to see the icons well enough to discern what they mean.

B. Property Inspectors

1. Introduction

Overcoming this problem, the present invention provides "property inspectors" for inspecting and setting the properties of objects. Property inspectors of the present invention examine an object of interest to the user and tell the user what can be done to it. In this manner, the present invention requires the system and its interface components (e.g., menus or dialog boxes), not the user, to know what functions are being sought or may be of immediate interest to the user. Moreover, property inspectors of the present invention require the program to present functions to the user when and where he or she requests them, rather than making the user hunt for them when they are needed. Thus, the user need only know which data item or object he or she wants to inspect or manipulate.

In the spreadsheet notebook, for example, there are many kinds of objects which one can use. Examples include cells, blocks of cells, pages, notebooks, graphs

and their elements (including bars or axes), dialog boxes, and even the application program itself. Each of these objects has properties, which are characteristics of that particular object. For example, blocks of cells have a font property that can be set to bold, so that the text of entries in the block appear in boldface type. A property of a workbook page, on the other hand, is the name that appears on its tab. Each notebook has its own palette property for controlling available colors. Application properties, for instance, include system defaults, such as a default storage directory or file extension. Thus in any system, the user has a variety of objects available, each of which has its own set of properties.

Property inspection of the present invention provides the user with immediate access to an object's properties. If the object of interest is a spreadsheet cell, for example, the property inspector of the present invention produces a menu that includes font, color, border, size, and other display properties, along with formula properties which may be modified. If, on the other hand, the object is a graph, the property inspector will display a menu to change its color, shape, borders, and other features of a spreadsheet graph. Moreover, inspection menus are state or context-sensitive, i.e., constructed on-the-fly to reflect the current state of the object. If an object's properties change so that what a user can do to it also changes, the property inspector of the present invention will create a new and appropriate menu reflecting those changes.

A variety of user events, including keyboard and mouse events, may be employed to invoke property inspection and setting. In a preferred method of the present invention, however, an object is inspected by selecting the object with a screen cursor, i.e., clicking (generating a mouse event) on the object. Since, according to the present invention, property inspection may be available across different modes of operation (i.e., generally available at all times), the generated mouse event or signal is preferably one which does not launch or otherwise invoke routine object actions. Instead, the mouse signal should be one which the user will easily associate with inspection of an object. In a two or three mouse button embodiment of the present invention, therefore, the generated mouse signal is most preferably from the lesser-used or right mouse button (e.g., Windows' WM_RBUTTONDOWN). In this manner, the user associates object actions with left button signals and object inspection with right button signals.

2. Exemplary Embodiments

Referring now to FIGS. 5A–E, the inspecting and setting of object properties, in accordance with the present invention, is illustrated. In FIG. 5A, a notebook application window **500** is shown. Window **500** includes a variety of objects which may be inspected by the property inspector of the present invention. The topmost object is the application itself: application object **510**. By selecting the application object (e.g., by right mouse clicking on the application title bar), various properties for the application may be inspected and set. As shown in FIG. 5B, inspection of the application object **510** invokes application inspector **515**, which lists the various properties of the application object; the user may now inspect and set the properties of the object, as desired.

Referring back to FIG. 5A, the next level which may be inspected is the notebook object **520**. User inspection of this object, e.g., by right mouse clicking on the notebook title bar, invokes the active notebook property inspector **525** of FIG. 5C. In a manner similar to the application property inspector, the notebook property inspector **525** includes all the properties and corresponding options which may be set for the notebook **520**.

Notebook object **520** includes a plurality of page objects, which may themselves be inspected. To inspect the page object **550**, for example, the user brings the page tab into view (if it is not already displayed), then right clicks the page tab. This action invokes the active page property inspector **535**, as shown in FIG. 5D. With access to page properties, the user can control the page name, overall page protection, line color, colors in cells that meet certain conditions, default label alignment, whether zeroes are displayed, default column width, row and column borders, gridline display, and the like. To assign a more descriptive name to a page, for example, its tab is inspected and the option "NAME" is selected from the page property inspector (or, as shown, it is selected by default). After the user enters a new name, the new name appears on the page tab. Other page properties, most of which affect the appearance of data or screen elements, may be set in a similar manner.

Each page object may also contain other objects, including cells, blocks of cells, graph objects, and the like, each of which may be inspected. By right clicking the active block **530**, the active block property inspector **545** is opened as shown in FIG. 5E. With block properties, the user can change alignment of entries, numeric format of values, block protection, line drawing, shading, font, text color, types of data allowed in a cell, row height, column width, and whether columns or rows are revealed or hidden; inspection of a single cell occurs in a similar manner (i.e., active block is one cell). In this manner, the user may select block properties which affect the appearance of cell entries or of rows or columns; additionally, properties which affect the way data can be entered into blocks, such as protection and data entry input, may be made.

Upon invocation of property inspection, a dialog box or property inspector is displayed, preferably at or near the object of interest. A different property inspector appears depending on the type of object which is inspected. At this point, the user may then select the property that he or she wishes to change from a list of object properties. The contents of the right side of the inspector (i.e., a "pane" within the inspector) change to correspond to the property one chooses, as shown. Then, the user chooses settings for the current property. If desired, the user can then choose and set other properties for the current object. In a preferred embodiment, the property inspector includes an example box, which shows the display result of the property the user has just set, before they are applied to the actual object.

Referring now to FIGS. 6A–K, the construction and operation of a property inspector dialog box, in accordance with the present invention, is described. For purposes of illustration, the following description will present active block property inspector **600**; other inspector dialogs may be implemented in a similar fashion. Shown in FIG. 6A, the active block property inspector **600** includes an object property list **601** and property settings pane **602**. For an active block, for example, relevant properties include Numeric Format, Font, Shading, Alignment, Line Drawing, Protection, Text Color, Data Entry Input, Row Height, Column Width, and Reveal/Hide. Property settings pane **602** include

the valid options or settings which a property may take. For the property of numeric format, for example, valid settings include fixed, scientific, currency, and the like. Property setting pane **602** may further include subproperties. For example, the currency setting **602***a* shown may also include a decimal place setting **602***b*. In this manner, property inspector dialog **600** presents a notebook dialog: property list **601** serves as notebook tabs (e.g., dialog tab **601***a*) and property settings panes **602** serves as different pages of the notebook dialog. Thus, the user may access numerous dialog options with the efficiency of a notebook.

Also shown, property inspector dialog **600** also includes an example window **605** and dialog controls **603**. As new properties are selected and set, their net effect is shown. The example element in example window **605**, for example, shows the effect of selecting a numeric format of currency with two decimal places. Thus, the user may experiment with changes before they are applied to the data object. After desired property changes are entered, control components **603** are invoked to either accept ("OK button") or reject ("CANCEL button") the new property settings; if desired, help may be requested.

As shown in FIGS. 6B–K, other properties of a cell block are accessed by selecting the desired property from the property list **601**. Thus the active blocks property inspector **600** changes to inspector **620** for font properties, inspector **625** for shading properties, inspector **630** for alignment properties, inspector **635** for line drawing properties, inspector **640** for cell protection properties, inspector **645** for text color properties, inspector **650** for data entry input properties, inspector **655** for row height properties, inspector **660** for column width properties, and inspector **665** for reveal/hide properties. In each instance, a new pane (i.e., one having the attributes of interest) is displayed in the inspector window.

Referring now to FIGS. 7A–H, the inspection (and setting) of properties for graphs is illustrated. Graph window **700** includes a plurality of graph objects, each of which may be customized through use of a corresponding property inspector of the present invention. To display a corresponding property inspector, the user invokes the inspector (e.g., right-clicks) for the part (object) of the graph he or she wishes to change. A right-click on the graph window object **710**, for example, will invoke the graph window inspector **715**; at this point, the user may inspect and set various properties of the graph window object. In a similar manner, other objects of the graph window may be inspected. For example, inspection of graph background **720** invokes inspector **725**, Y-axis object **730** invokes inspector **735**, X-axis **740** invokes inspector **745**, area fill object **750** invokes inspector **755**, bar series object **760** invokes inspector **765**, and bar series object **770** invokes inspector **775**.

Internal Operations

A. Introduction

Internal operations of the system **100** will now be described in detail. In general, operation is driven by methods of the present invention, which are invoked by Windows' message dispatcher in response to system or user events. The general mechanism for dispatching messages in an event-based system, such as Windows, is known in the art; see, e.g., Petzold, C., *Programming Windows*, Second Edition, Microsoft Press, 1990. Additional information can be found in Microsoft's *Window Software Development Kit*, including: 1) Guide to Programming, 2) Reference, Vols. 1 and 2, and 3) Tools, all available from Microsoft Corp. of Redmond, Wash. Of particular interest to the present invention are class hierarchies and methods of the present invention, which are implemented in the C++ programming language; see e.g., Ellis, M. and Stroustrup, B., *The Annotated C++ Reference Manual*, Addison-Wesley, 1990. Additional information about object-oriented programming and C++ in particular can be found in Borland's C++ 3.0: 1) *User's Guide*,2) *Programmer's Guide*, and 3) *Library Reference*, all available from Borland International of Scotts Valley, Calif. The disclosures of each of the foregoing are hereby incorporated by reference.

B. Notebooks

Referring now to FIG. 8A, the internal structure and operation of spreadsheet notebooks of the present invention will be described. The spreadsheet notebook system of the present invention includes a system hierarchy **800**, at the top level, having a notebook table **805** with entries or slots for accessing a plurality of notebooks. For example, the first slot addresses or points to notebook **810**. In this manner, the system may track multiple notebooks.

Notebook structure **810**, in turn, includes or accesses various data members for a particular notebook. For example, a "Name" field, stores the name assigned for the notebook. This name is displayed in the titlebar for the notebook and is used as the name for the corresponding disk file; the notebook name is also used to reference the notebook in formulas (e.g., contained in other notebooks). Notebook **810** also includes other data members, such as block names and fonts. Block names are text strings or labels which the user has defined for particular blocks of interest. Fonts, on the other hand, include font information (e.g., type and size) for the notebook. Other desired properties, specific to a notebook, may be included as desired. As exemplary construction of a notebook structure (class), in accordance with the present invention, is set forth hereinbelow in Appendix A.

Of particular interest in the notebook structure **810** is the Pagetable field, which includes a pointer to a page table for the notebook. Pagetable **815** includes a plurality of slots or entries for accessing individual page structures of the notebook. Each page structure, in turn, contains or accesses information of interest to an individual page. As shown, for example, page **820** (pointed to by the first slot of the pagetable) includes or accesses: a sheet table (or Graphs page), a pagename, floating objects (e.g., graphs and buttons), page properties, pane, and the like.

The Sheettable field of the page **820** points to a sheet table **825**. It, in turn, includes a plurality of the slots for accessing different column strips. As shown, for example, the first entry in the sheet table **825** accesses a column strip of cells **830**. In turn, column strip **830** addresses individual information cells, each of which may have one of a variety of information types, including empty, integer, number (real), formula, label, and the like. In a preferred embodiment, column strip **830** may address up to 8,000 cells; those skilled in the art, however, will appreciate that column strip **830** may be set to any desired range (depending on the limits of one's target hardware).

Referring now to FIGS. 8B–C, the function of the page table of the present invention is illustrated. In FIG. 8B, two images are presented: a view image and a data

image. The view image illustrates what the user sees on the screen **106**; at startup, for example, only a single Page A (page **850**) is active. As shown by the corresponding data image supporting this view image, pagetable **855** includes only a single reference, page A. In turn, page A references sheettable$_A$, which supports the information cells required (e.g., they are currently needed for display) for the page. Thus, at startup, only a single page entry exists in the pagetable **855**, even though the screen displays multiple page tabs.

Upon selection of Page F (e.g., by clicking tab F **860**), the data image changes, however. As shown in FIG. **8C**, remaining Pages B–F are initialized into the page table **865**. In this example, Page F must now exist as a view object. Thus, Page F references supporting data structures (e.g., sheettable$_F$). The intervening Pages (B–E), on the other hand, are initialized, but only to the extent that they can serve as place holders in the pagetable **865**. In this manner, the underlying page table for a notebook instantiates underlying page data structures only as needed (e.g., for viewing or referencing by other pages), but, at the same time, provides on-demand access to pages.

A particular advantage of this design is the ease in which information may be referenced and presented. Even complex data models, such as those spanning multiple dimensions, may be represented in a clear fashion. Moreover, a syntax is provided by the present invention for intuitively referencing information.

Referring now to FIGS. **8D–F**, the referencing of information in a spreadsheet notebook of the present invention is now described. Shown in FIG. **8D**, a three-dimensional reference (i.e., identifier for a solid block of information cells) includes a notebook, starting page, ending page, starting cell and ending cell. As shown, the notebook (which is the same as the file name) is identified preferably by delimiters, such as brackets ([]). This is followed by page name(s), which may in fact be grouped. As shown, a range of pages has been defined from '89 Income to '92 Income; these are alias names which may correspond to pages A–D, for example.

Next, one or more cells of interest are identified. For example, a range from January to (. . .) December is shown, which serve as aliases for blocks **A1** to **A12**, for example. The block or cell information is separated from group or page information, for example, by a colon (:) identifier.

The hierarchy of this nomenclature is shown in FIG. **8E**. Specifically, a notebook references pages, which may be members of one or more user-defined groups. In turn, a page references cells, which may alternatively be referenced by alias identifiers. As shown in FIG. **8F**, page and cell identifiers may be grouped to form even more succinct references. For example, Pages '89 Income to '92 Income may be renamed as Four-Year Income. Similarly, the cell range from January to December may be renamed Entire Year. In this manner, information ranges in the spreadsheet notebook of the present invention are easily named and easily visualized by the user.

Depending on the context of the system, certain identifiers may be assumed and, thus, eliminated from a reference. Unless otherwise specified, for example, the notebook identifier is assumed to be the currently active notebook. Similarly, the page identifier may be assumed to be the currently active page, unless the user specifies otherwise. Thus, a valid reference need only include as much information (identifiers) as is necessary to access the desired information.

Referring now to FIG. **9A**, a method for interpreting or parsing an information reference, in accordance with the present invention, is shown. Upon receiving a reference (e.g., text string) to information in a spreadsheet notebook, the method proceeds as follows. In step **901**, a loop is established to tokenize (i.e., parse) the string into individual identifiers, which may then be processed. Each token is examined as follows. In step **902**, the token is checked to determine whether it is a notebook identifier. In a preferred method, a notebook identifier is delimited, for example, by bracket symbols; alternatively, a notebook identifier may be realized simply by the position of the identifier relative to other tokens in the string, or by its relationship to disk (notebook) files on mass storage **107**. If the notebook is successfully identified, the notebook identifier is returned at step **903**, and the method loops back to step **901** for any additional tokens.

If it is not a notebook (no at step **902**), however, then in step **904** the token is examined to determine whether it is a group name. Group names may be determined by accessing group names listed for the notebook (e.g., by accessing a group name string pointer from notebook structure **810**), and/or by the token's context (e.g., preceding a ":" delimiter). If a group name is identified, it is returned at step **905**, and the method loops for any remaining tokens. Otherwise (no at step **904**), the token is examined to determine whether it is a page. In a manner similar to that for checking group names, a page may be identified by examining a notebook's page table, with corresponding page name accessed (dereferenced). The page is returned at step **907**, with the method looping back to step **901** for remaining tokens.

If a page is not found (no at step **906**), however, then at step **908** the token is examined to determine whether it defines a range. A range may include a named block of cells (e.g., "entire year") or, simply, two cell addresses separated by an appropriate identifier (e.g., **A1** . . . **B1**). If a range is found, then it is returned in step **909**, with the method looping for any remaining tokens. Otherwise (no at step **908**), the identifier is examined to determine whether it is a cell. A token may be identified as a cell if it is of the appropriate column/row format (e.g., A1). If a cell is found, it is returned at step **911**, with the method looping for any remaining tokens.

As shown (conceptually) at step **912**, if any ambiguities remain, they may be resolved according to an order of precedence; for example, notebook >groupname>page and the like. At the conclusion of the method, a reference, if it is in proper form, will be successfully identified and may now be processed according to three-dimensional techniques.

Referring now to FIGS. **9B–C**, other methods of the present invention are illustrated. In FIG. **9B**, for example, a drag-and-drop edit method **920** of the present invention is shown. Its steps are as follows. In step **921**, a block of cells is defined (e.g., by dragging a mouse cursor from one corner of the block to another). After a block has been selected, it is next grabbed with the mouse cursor (i.e., position the mouse cursor within the block and depress a mouse button). In steps **923** and **924**, a loop is established to "drag" the block to a new location. In particular, while the mouse button is depressed, any movement of the mouse will cause the block to follow (animated on screen, e.g., by XOR technique). At step **925**, the block is "dropped" into place by releas-

ing the mouse button (button up signal). Dropping the block causes information from the source location to be copied into the target location. By coupling the method with a status key (e.g., SHIFT- or CTRL-), the technique supports either "cut" (erase original) or "copy" (leave original) modes of operation. Thus, for example, if a status key is not active as step **926**, then in step **927** the original (at the source location) is deleted. Otherwise (yes at step **926**), the original remains at the source location as well.

Referring now to FIG. 9C, a model copy method **940** of the present invention is illustrated. In step **941**, a block is defined or selected (e.g., dragging a selection). In step **942**, model copy is enabled or disabled (as desired); alternatively, model copy may be enabled by default. In step **943** if model copy has been enabled, then in step **945** absolute address references are copied as if they were relative address references, as previously described (with reference to FIGS. 4H–J). However, the address labels will remain absolute, so that they will be treated as absolute for future copying operations. Otherwise (no at step **943**), absolute addresses are treated conventionally (i.e., referencing absolute addresses) in step **944**. As shown in step **946**, relative addresses are not affected, i.e., they continue to be treated relatively. In step **947**, the copy operation is performed, employing the addresses as just determined, after which the method concludes.

Additional methods of the present invention are set forth hereinbelow in Appendix A

C. Property Inspection

Referring now to FIGS. 10A–C, construction and operation of property inspection in accordance with the principles of the present invention will be described. As shown in FIG. **10A**, user interface (UI) objects are constructed from a UI object class hierarchy **1000**. As shown, class hierarchy **1000** includes as its base class an object class **1001**. From object class **1001**, a responder class **1002** is derived. As the child of class **1001**, responder class **1002** inherits all properties of its parent and adds event handling capability. Both object class **1001** and responder class **1002** are abstract classes, i.e., objects are not instantiated from them directly. From responder class **1002**, two child classes are derived: view class **1003** and window class **1004**. From view class **1003**, all screen objects (e.g., text, graphs, scrollbars, and the like) are created. From window class **1004**, container objects are instantiated; in particular, window class **1004** provides container objects which hold the various view objects.

Associated with each class (and thus objects derived from each class) is a ClassInfo structure. ClassInfo, which is itself implemented as a separate class, contains information about a corresponding class and, thus, objects of that class. For example, it contains information about object type, name, number of properties, and the like. Of particular relevance to property inspection are two data members: a pointer to the parent and a property record pointer which points to an array of property records for an object of interest.

Referring now to FIG. **10B**, the relationship between parent and child (and hence the importance of the pointer to the parent) is illustrated. In the system of the present invention, an object-oriented mechanism is employed whereby new objects may be created from existing objects (classes). As shown, for example, a child object **1030** may be derived from a parent object **1020**. Also shown, each object has its own property record array or list. For example, the parent object **1020** has a property list **1025** describing its properties. Child object **1030**, in turn, also has its own property list **1035**; child object **1030** still inherits from parent object **1020**, however. By means of the parent pointer, the child object **1030** also has direct access to its parent **1020** and, thus, the property list **1025** of the parent. In this manner, when an object is inspected, the system of the present invention may view back across the inheritance hierarchy to fetch all (ancestor) properties for the object, as needed.

The property record, on the other hand, describes an individual property for the object. The property record, which is implemented as a separate class, includes a name ID for the property and a pointer to the property object itself (which may be shared). For example, property record objects may be instantiated from the following exemplary C++ class definition:

```
class _EXPORT_ PropRecord
{
public:
    Property * pProp;          // pointer to SHARED (!) property object
    WORD       nameID;         // property name string ID
                               // (PROPSTR.HPP, PROPSTR.CPP)
    WORD       flags;          // optional information about the property
                               // ( HIDDEN, USE MENU, LINK ONLY, etc.)
    inline Property * getProp ();
    inline LPSTR getName ();
    inline PROPTYPE getType ();
    inline WORD getNamesCnt ();
    inline WORD getNameID ();
    inline WORD getFlags ();
};
```

The property object, in turn, includes a type ID for identifying the property and also includes a pointer to a dialog for the property. An property object may be instantiated from the following exemplary C++ class:

```
class _EXPORT_ Property
{
public:
    static DWORD dwPropErr;
    static char far conversionBuffer [MAXPROPSTR];
    PROPTYPE typeID;
```

-continued

```
    WORD namesCnt;
    LPSTR pDialog;
    Property ( WORD id );
    virtual BOOL stringToValue ( LPSTR, PROPTYPE pt = 0 );
    virtual BOOL valueToString ( LPSTR, PROPTYPE pt = 0 );
/*
Convert the passed binary value to string using the conversionBuffer
*/
    LPSTR convertToString ( LPSTR pS, PROPTYPE pt );
/*
Convert the string in conversionBuffer to binary value pointed by pS
*/
    BOOL convertToBinary ( LPSTR pS, PROPTYPE pt );
};
```

Exemplary property types may include:

prop_Undefined=0,
prop_Text,
prop_Bool,
prop_Window,
prop_Color,
prop_Bitmap,
prop_List,
prop_Word,
prop_Int,
prop_Key,
prop_Double,
. . .

The pointer to the property dialog (LPSTR pDialog), on the other hand, indicates the appropriate dialog to be displayed to the user for the current property; the dialog displays the appropriate information (e.g., pane) for the property which the user may then use for inspecting and setting attributes of that property.

Referring now to FIG. **10C**, a method **1050** for property inspection, in accordance with the present invention, is illustrated; additional disclosure is provided for the inspection of an exemplary object: a spreadsheet cell. In step **1051**, the user indicates that he or she desires property inspection for an object. In a preferred method of the invention, the user will right-mouse click on the object of interest (as set forth hereinabove). To inspect a spreadsheet cell, for example, the user would position the mouse cursor over the cell and click the right mouse button. In this instance, the notebook which contains that cell will receive (i.e., will be the recipient of) a right-mouse button down message. The event will be processed by routing the message to a RightMouseDown method.

In step **1052**, the object which the user clicked on is identified by the RightMouseDown method. Specifically, the method invokes a RouteMouseEvent method which returns the current view object (i.e., the object which the user has clicked on). This is accomplished as follows. Every view object contains a StartInspect method which returns an actual object for inspection. The object which appears on the screen to the user (i.e., view object) is not necessarily the object which will be inspected. For example, it would be computationally too expensive to have each cell of a spreadsheet be its own object. Instead, the present invention embraces the notion of surrogate or imaginary objects which may be inspected (using the properties of the screen or view object selected). In most instances, the StartInspect method will return an object for inspection which is the same as the view object. In those instances where it is not feasible to inspect an object which the user perceives on a screen, however, the method returns a surrogate object whose properties assume the attributes of the screen object.

An additional example illustrates this point. When inspecting a block of cells, for example, StartInspect returns a current block object which is a surrogate object, i.e., it does not have a visible expression on the screen. Instead, it is a substitute object which assumes the characteristics necessary to allow inspection of the screen object of interest. In this manner, it functions as a surrogate for the object which the user observes on the screen. If desired, objects may also be blocked from inspection at this step (e.g., by setting flags); in which case, the method terminates. At the conclusion of step **1052**, the object of interest, either actual or surrogate, is ready for inspection.

Next, the user interface for inspection is built by a DoUI method of the present invention. The method proceeds as follows. The first property record for the object is accessed in step **1053**. Preferably, DoUI is a virtually defined method, with each object (class) designed to include method steps for accessing the property record according to that object's own particularities. The remaining property records for the object are also located (e.g., by traversing a list of records).

At step **1054**, the dialog panes for each property are loaded (e.g., into memory **102**, if not already present) for use by a compound dialog box. As previously described, the dialog associated with each property is accessible from the property record (by a pointer to the dialog). At this point, an empty property inspection window is displayed. Into the property inspection window, a corresponding pane for each property is loaded (substep **1054***b*), thus creating a display hierarchy. Using a getProperty method, each property is initially set to the attribute(s) contained in the retrieved property record; the getProperty method operates essentially the reverse of a setProperty method, which is described in detail hereinbelow. An associated screen button or label is provided for rapidly accessing a desired pane in the inspector window, in much the same fashion as one accesses pages in a notebook with tabs. In this manner, a property inspector is built from a dynamic list of properties (as opposed to a static list of properties which is determined beforehand), each of which may be rapidly accessed by the user.

While the property list loaded into an inspector window for each object is dynamically constructed, the panes supporting each property may be pre-built. For example, the pane supporting a color palette, since it is complex in design, will be built in advance.

However, the method of the present invention may dynamically build individual panes as well. Candidates for dynamic pane building include simple properties,

especially those which may have only a Boolean or yes/no value. Referring back to FIG. 7B, for example, inspector **715** includes a snap-to-grid property. Instead of loading a preconstructed pane for this property, the method of the present invention dynamically constructs an appropriate pane (in this case, a simple check box), as shown in substep **1054a**. An automatic list, on the other hand, is typically a simple group box (i.e., one having a plurality of radio buttons), which is preferably constructed dynamically, as shown by the inspector window **650** of FIG. **6H**. In either case, the method may build appropriate inspector dialog components on-the-fly by simply examining a particular property and discerning its possible attributes. In a similar manner, "pick" lists of properties may be constructed and displayed, as shown in substep **1054c**. A property pick list serves as an index to other property lists or other functions, as desired. By dynamically building inspectors for simpler properties, the method conserves system resources.

Construction of the inspector window is completed by inserting dialog controls (e.g., "OK", "CANCEL", and "HELP") for operating the dialog. In addition, an example window is displayed for indicating various changes to the properties of an object. This is accomplished by sending a message to the inspected object setting forth the current properties in the dialog; the inspected object returns an example or sample object which may then be displayed in the window. After changing a property, the dialog tab or button (e.g., tab **601***a* of FIG. **6A**) corresponding to that property is updated (e.g., different font or screen color) for indicating that a change has been entered.

After constructing the property inspector dialog or window, at step **1055** the method enters a user event loop where the user inspects and sets various properties of the object by accessing the property (through the screen button or tab) and setting a new attribute. At the conclusion of the user event (e.g., by selecting "OK" or "CANCEL"), the user event is terminated.

At step **1056**, the property list for the object being inspected is updated for any changes which occurred to the properties during the user event (of step **1055**). This is accomplished by calling a setProperty method, which conceptually loops through each pane which has changed and updates the property list, accordingly. By way of illustration, the setProperty method may be defined as:

```
virtual BOOL setProperty(LPSTR lpPropStr,
    LPSTR lpValueStr, PROPTYPE pt=0);
```

The method receives the name of a property, either passed as a text string (lpPropStr) or as a handle (pt), and a value for that property, typically passed as a text string (lpValueStr).

Referring now to FIG. **10D**, the general steps of a setProperty method **1070** are illustrated. In step **1071**, the property text string is translated[; alternatively, the property is referenced directly by passing a handle (i.e., integer or word value). At steps **1072–1073**, the property is updated for the property value passed (e.g., by a CASE statement construct). If the property is not acted upon, it may be passed up the object's inheritance chain for action by an ancestor object, at step **1074**. In this fashion, the values collected from the various property panes are used by the method to set the properties within the object.

While the foregoing outlines the general steps or framework for setProperty, the method is in fact defined as a virtual method. Thus, each class of objects may assume responsibility for setting its own properties. For an Annotate object (i.e., screen objects found in Graph windows), for example, an exemplary setProperty method may be constructed as follows:

```
BOOL Annotate::setProperty ( LPSTR lpPropStr, LPSTR lpValueStr, PROPTYPE
pt )
{
    WORD w;
    if ((w = info.propIndex (lpPropStr, pt)) < annoEndProps) {
        if (info.ppProps [w].getType() > prop_Compound)
            switch (w) {
                case annoFrame:
                    getFrame (GlobalFrameProp.x,
GlobalFrameProp.y,
                        GlobalFrameProp.w,
GlobalFrameProp.h);
                    break;
            }
        if ((info.ppProps [w].getProp()->stringToValue (lpValueStr,
pt)) {
            BOOL f = TRUE;
            switch (w) {
                case annoFrame:
                    setFrame (GlobalFrameProp.x,
GlobalFrameProp.y,
                        GlobalFrameProp.w,
GlobalFrameProp.h);
                    break;
                case annoHidden:
                    if (GlobalYesNoProp.index)
                        hide (TRUE);
                    else
                        show (TRUE);
                    break;
                case annoShow:
                    if (GlobalYesNoProp.index)
                        show (TRUE);
                    else
                        hide (TRUE);
                    break;
```

-continued

```
                case annoId:
                    if (pCW && pCW->infoPtr()->objType ==
ot_Dialog)
                        f = ((DialView
*)pCW->getContentView())->setIdx (this, GlobalWordProp.w);
                    else
                        idx = GlobalWordProp.w;
                    break;
                case annoAttach:
                    flags.nochild = !GlobalYesNoProp.index;
                    break;
                case annoPosition:
                    *(WORD *)&flags = (*(WORD *)&flags
                        & ~(vfPOSITION | vfLEFTREL |
vfTOPREL | vfRIGHTREL | vfBOTTOMREL
                            | vfTOPTYPE | vfRIGHTTYPE)) |
GlobalPositionProp.flags;
                    flags.bottomType = flags.topType;
                    flags.leftType = flags.rightType;
                    break;
            }
            if (f)
                callConnection (lpPropStr, lpValueStr, pt);
            return f;
        }
        else
            return FALSE;
    }
    else
        return Tracker::setProperty (lpPropStr, lpValueStr, pt);
}
```

As shown, the override method processes properties specific for Annotate (e.g., change frame, ID, position, and the like). In the event that a property is not identified, the property is passed up the inheritance chain for appropriate processing (by a parent's setProperty method). In this manner, an individual object (e.g., an Annotate object) in the system is designed to know how to get (getProperty) and set (setProperty) its own properties.

After the update of step **1056**, the method **1050** concludes. At this point, the system is ready for another inspection. Alternatively, the user may undertake other activities, as desired.

Attached hereto is a microfiche Appendix A containing C++ source code listings, which provide a description of the invention suitable for use in a general purpose digital computer system, such as an IBM-compatible personal computer. A suitable compiler for compiling and linking C++ code is available from Borland International.

While the invention is described in some detail with specific reference to a single preferred embodiment and certain alternatives, there is no intent to limit the invention to that particular embodiment or those specific alternatives. For example, the property inspection of the present invention has been illustrated with mouse devices. Those skilled in the art, however, will appreciate that other input devices, including trackballs, joysticks, light pens, and the like, may be employed. Moreover, the present invention may be advantageously implemented in a variety of UI platforms other than Windows, including Macintosh, X-Windows, Motif, NextStep, OS/2, and the like. Thus, the true scope of the present invention is not limited to any one of the foregoing exemplary embodiments but is instead defined by the following claims.

What is claimed is:

1. In an electronic spreadsheet system for storing and manipulating information, a method of representing and using a three-dimensional spreadsheet on a screen display, the method comprising:

(a) displaying on said screen display a first spreadsheet page from a plurality of spreadsheet pages, each of said spreadsheet pages comprising an array of information cells arranged in row and column format, at least some of said information cells storing user-supplied information and formulas operative on said user-supplied information, each of said information cells being uniquely identified by a row, a column, and a spreadsheet page where the cell is located;

(b) while displaying said first spreadsheet page, displaying a row of spreadsheet page identifiers along one side of said first spreadsheet page, each said spreadsheet page identifier being displayed as an image of a notebook tab on said screen display and indicating a single respective spreadsheet page; and

(c) designating a three-dimensional block of cells in said three-dimensional spreadsheet by the steps of:

i) using an input device to designate a two-dimensional block of cells on said first spreadsheet page; and

ii) designating a selected one of said notebook tabs to designate a second spreadsheet page, and extend said two-dimensional block to a three-dimensional block of cells

2. The method of claim **1** wherein:

step (b) further includes displaying a page identifier for said first spreadsheet page

3. The method of claim **1**, further comprising:

(d) while displaying said first spreadsheet page, indicating on said screen display that the spreadsheet page being displayed is said first spreadsheet page.

4. The method of claim **3** wherein step (d) comprises displaying on said screen display a spreadsheet page identifier for said first spreadsheet page with a different appearance from other displayed spreadsheet page identifiers.

5. The method of claim 4 where said different appearance includes a difference in color.

6. The method of claim 1, further comprising:

(d) selecting with said input device a spreadsheet page identifier for a third spreadsheet page; and

(e) in response to step (d), displaying said third spreadsheet page while simultaneously displaying said spreadsheet page identifiers for said first and second spreadsheet pages, said third spreadsheet page obscuring from display said first and second spreadsheet pages.

7. The method of claim 1, wherein said spreadsheet page identifiers are displayed along a horizontal side of a spreadsheet page being displayed.

8. The method of claim 7 wherein said horizontal side is a bottom side.

9. The method of claim 1, wherein each spreadsheet page identifier includes a default page name comprising a single alphabetic character.

10. The method of claim 1, wherein a cell of one spreadsheet page includes a reference to a cell of another spreadsheet page, said reference including a page name for said other spreadsheet page.

11. The method of claim 1, further comprising:

(d) receiving a request for changing a default page name associated with one of said spreadsheet page identifiers; and

(e) in response to step (d), changing said default page name associated with, said one of said spreadsheet page identifiers to a user-supplied page name.

12. The method of claim 1, wherein said user-supplied page name comprises textual information.

13. The method of claim 1, wherein a reference to an information cell located on a spreadsheet page having a user-supplied page name includes said user-supplied page name as a portion of the reference.

14. The method of claim 1, wherein:

step (b) includes displaying a spreadsheet page identifier for said first spreadsheet page and providing said spreadsheet page identifier for said first spreadsheet page a different appearance than remaining spreadsheet page identifiers in said row of spreadsheet page identifiers.

15. The method of claim 1, further comprising:

(d) receiving a request for moving a particular one of said spreadsheet pages to a desired position relative to others of said spreadsheet pages, said request comprising a user action which specifies said desired position by positioning the spreadsheet page identifier for said particular one of said spreadsheet pages at a new position relative to other spreadsheet page identifiers; and

(e) in response to step (d), moving said particular one of said spreadsheet pages to said desired position.

16. In an electronic spreadsheet system for storing and manipulating information, a method of representing and using a three-dimensional spreadsheet on a screen display, the method comprising:

(a) displaying on said screen display a first spreadsheet page from a plurality of spreadsheet pages, each of said spreadsheet pages comprising an array of information cells arranged in row and column format, at least some of said information cells storing user-supplied information and formulas operative on said user-supplied information, each of said information cells being uniquely identified by a row, a column, and a spreadsheet page identifier;

(b) while displaying said first spreadsheet page, displaying a row of spreadsheet page identifiers along one side of said first spreadsheet page, each said spreadsheet page identifier being displayed as an image of a notebook tab on said screen display and indicating a single respective spreadsheet page, wherein:

i) each of said notebook tabs comprises at least one user-supplied identifying character, said user-supplied identifying character used as said spreadsheet page identifier for cells on spreadsheet pages associated with their respective notebook tabs, each of said plurality of said spreadsheet pages in said three-dimensional spreadsheet associated with a unique one of said spreadsheet page identifiers; and

ii) said user-supplied information in said rows and columns of information cells for each of said plurality of spreadsheet pages is stored in a single disk file;

(c) requesting display of a second spreadsheet page in response to selection with an input device of a spreadsheet page identifier for said second spreadsheet page;

(d) in response to said requesting step, displaying said second spreadsheet page from said single disk file on said screen display in a manner so as to obscure said first spreadsheet page from display while continuing to display at least a portion of said row of spreadsheet page identifiers;

(e) using said spreadsheet page identifiers to identify a cell in a desired spreadsheet page when entering said formulas; and

(f) designating a three-dimensional block of cells in said three-dimensional spreadsheet by the steps of:

i) using an input device to designate a two-dimensional block of cells on said first spreadsheet page; and

ii) designating a selected one of said notebook tabs to designate a second spreadsheet page, and extend said two-dimensional block to a three-dimensional block of cells.

17. In an electronic spreadsheet system for modeling user-supplied information in a three-dimensional spreadsheet model, said three-dimensional spreadsheet model comprising a plurality of information cells storing data and formulas operative on said data, said system performing an aggregate operation on selected blocks of cells, a method for assisting a user with selecting a three-dimensional block of cells, the method comprising:

(a) dividing said three-dimensional matrix of information cells into a plurality of two-dimensional matrices, each of said two-dimensional matrices being represented on a display device as a page array of information cells, wherein cell references between cells on one page array are formed by specifying row and column coordinates, and cell references between cells on different page arrays are formed by specifying row, column, and page coordinates, said three-dimensional matrix of information cells being stored on a storage device as a single disk file;

(b) displaying with each said page array of information cells a tab identifier, so that each tab identifier of a particular page array displays the page coordinate for that particular page array;

(c) receiving first user input for selecting a two-dimensional block of cells from a first page array,

whereupon said system selects said two-dimensional block of cells from said first page array;

(d) while said two-dimensional block of cells is selected, receiving second user input for extending said two-dimensional block of cells into a three-dimensional block of cells, said second user input including user selection of the tab identifier for a second page array, whereupon said system extends said two-dimensional block of cells from said first page array into said three-dimensional block of cells spanning from said first page array to said second page array; and

(e) receiving third user input for requesting said aggregate operation, whereupon said system performs said aggregate operation on said three-dimensional block of cells.

* * * * *

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.
08/316,237	09/30/94	CHRISTENSEN S	04860.P1365

24M1/1120

BLAKELY SOKOLOFF TAYLOR AND ZAFMAN
12400 WILSHIRE BOULEVARD
7TH FLOOR
LOS ANGELES CA 90025

EXAMINER	
DELA TORRE, C	
ART UNIT	**PAPER NUMBER**
2415	9

DATE MAILED: 11/20/96

Please find below and/or attached an Office communication concerning this application or proceeding.

Commissioner of Patents and Trademarks

PTO-90C (REV. 2/95) 08/316 237

1 - File Copy

Office Action Summary	Application No. 08/316,237	Applicant(s) Christensen
	Examiner Crescelle Dela Torre	Group Art Unit 2415

☒ Responsive to communication(s) filed on *Aug 23, 1996* .

☒ This action is **FINAL.**

☐ Since this application is in condition for allowance except for formal matters, **prosecution as to the merits is closed** in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11; 453 O.G. 213.

A shortened statutory period for response to this action is set to expire *three* month(s), or thirty days, whichever is longer, from the mailing date of this communication. Failure to respond within the period for response will cause the application to become abandoned. (35 U.S.C. § 133). Extensions of time may be obtained under the provisions of 37 CFR 1.136(a).

Disposition of Claims

☒ Claim(s) *1-24* is/are pending in the application.

Of the above, claim(s) _____ is/are withdrawn from consideration.

☐ Claim(s) _____ is/are allowed.

☒ Claim(s) *1-24* is/are rejected.

☐ Claim(s) _____ is/are objected to.

☐ Claims _____ are subject to restriction or election requirement.

Application Papers

☐ See the attached Notice of Draftsperson's Patent Drawing Review, PTO-948.

☐ The drawing(s) filed on _____ is/are objected to by the Examiner.

☐ The proposed drawing correction, filed on _____ is ☐ approved ☐ disapproved.

☐ The specification is objected to by the Examiner.

☐ The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. § 119

☐ Acknowledgement is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d).

☐ All ☐ Some* ☐ None of the CERTIFIED copies of the priority documents have been

☐ received.

☐ received in Application No. (Series Code/Serial Number) _____ .

☐ received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

*Certified copies not received: _____ .

☐ Acknowledgement is made of a claim for domestic priority under 35 U.S.C. § 119(e).

Attachment(s)

☒ Notice of References Cited, PTO-892

☒ Information Disclosure Statement(s), PTO-1449, Paper No(s). *9*

☐ Interview Summary, PTO-413

☐ Notice of Draftsperson's Patent Drawing Review, PTO-948

☐ Notice of Informal Patent Application, PTO-152

--- SEE OFFICE ACTION ON THE FOLLOWING PAGES ---

Part III DETAILED ACTION

1. This action is responsive to communications: Amendment A, filed on 8/23/96. This action is made final.

2. Claims 1 - 24 are pending in this case. Claims 1, 11, 15 are independent claims. In Amendment A, filed on 8/23/96, claims 1, 5 - 12, 15 were amended, and claims 19 - 24 were added.

3. The present title of the invention is "Method and Apparatus for Displaying and Accessing Control and Status Information in a Computer System" as originally filed.

Claim Objections

4. Claims 15 - 18 are objected to because of the following informalities: in claim 15, lines 3, "create" should be --creating--. Appropriate correction is required.

Claim Rejections - 35 USC § 102

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. § 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless --

(a) the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for a patent.

6. Claims 1 - 3, 8 - 24 are rejected under 35 U.S.C. § 102(a) as being anticipated by Cohausz (EPO 0 584 392 A1), based upon the English translation, which is included with this Office Action.

As per claim 1, Cohausz teaches an "interactive computer-controlled display system" with a 'status indicator for a computer program', that comprises:

"a processor" which is inherently taught by Cohausz, since a processor is necessary in order to execute the functions of the status indicator;

"a data display screen" with 'monitor screen' at p. 4, paragraph 5;

"a cursor control device" with a 'mouse cursor' at p. 5, paragraph 2;

"a window generation and control logic" to "create an operating environment for a plurality of individual programming modules that provide status and control functions" at the bridging paragraph of pp. 2 - 3, "wherein the window generation and control logic generates and displays a first window region" with oblong field 1, at Figs. 1 - 3, and at p. 4, paragraph 5, "having a plurality of display areas" with individual fields 2, at Fig. 1, and at p. 4, paragraph 5, "wherein each of the plurality of display areas is associated with one of the plurality of individual programming modules" at p. 3, paragraph 2;

"an indicia generation logic" to "execute at least one of the plurality of programming modules to generate information for display in one of the plurality of display areas" at p. 3, paragraph 2, "wherein at least one of the plurality of display areas and its associated programming module is sensitive to user input" at p. 5, paragraph 2, and further using "message-based communication to exchange information to coordinate activities of the indicia generation logic to enable interactive display activity" at p. 3, paragraph 2, which teaches information passing between the status indicator and the respective program area, text, or information segment.

As per claim 2, Cohausz teaches a "control strip" with oblong field 1, at Figs. 1 - 3. Regarding claim 3, Cohausz also teaches that "at least one display area is variably sized" at p. 5, paragraph 1, and p. 6, paragraph 2.

In addition, Cohausz teaches that "at least one of the plurality of display areas only displays information" [claim 8] at p. 3, paragraph 2, and at Figs. 1 - 3; or "acts to provide access to control information when selected" [claim 9] at p. 3, paragraph 2, or "displays an additional display element" [claim 10] at p. 6, paragraph 3.

In reference to claim 11, Cohausz teaches the following subject matter:

"a processor" which is inherently taught by Cohausz, since a processor is necessary in order to execute the functions of the status indicator;

"a data display screen" with 'monitor screen' at p. 4, paragraph 5;

"a cursor control device" with a 'mouse cursor' at p. 5, paragraph 2;

"a window generation and control logic" to "create an operating environment for a plurality of individual programming modules that provide status and control functions" at the bridging paragraph of pp. 2 - 3, "wherein the window generation and control logic generates and displays a first window region" with oblong field 1, at Figs. 1 - 3, and at p. 4, paragraph 5, "having a plurality of display areas" with individual fields 2, at Fig. 1, and at p. 4, paragraph 5, "wherein each of the plurality of display areas is associated with one of the plurality of individual programming modules" at p. 3, paragraph 2;

"at least one indicia graphics generation logic" that "generates user sensitive graphics for display in at least one data display area by executing at least one of the plurality of programming modules" at p. 3, paragraph 2; and

wherein the window generation and control logic determines when a data display area has been selected, signals the indicia graphics generation logic, which then initiates a response from said at least one of the plurality of programming modules, also at p. 3, paragraph 2.

Cohausz also teaches that the "first window region is always visible to the user" [claim 12] at p. 4, paragraph 5, since he teaches that the status indicator is 'visible during the entire program'. As per claims 13, 14, they correspond respectively to claims 2, 3.

As to claim 15, Cohausz teaches the following steps:

"creating an operating environment for a plurality of individual programming modules that provide status and control functions" at p. 2, paragraph 4 to p. 3, paragraph 2;

"generating a first window" with oblong field 1, at Figs. 1 - 3, and at p. 4, paragraph 5, to accommodate a "plurality of display areas for indicia" with individual fields 2, at Figs. 1 - 3, and at p. 4, paragraph 5, resulting from "executing at least one of the plurality of individual programming modules, wherein each of the plurality of display areas is associated with one of the plurality of individual programming modules" at p. 3, paragraph 2;

"displaying an indicia" as shown at Figs. 1 - 3;

"selecting one of the indicia" at p. 5, paragraph 2; and

"said programming module performing a function in response to the selection" at p. 5, paragraph 2.

In addition, Cohausz teaches "status information" [claim 16] and "control information" [claim 17] at Figs. 1 - 3, and at p. 5, paragraphs 2, 3.

As to claim 18, Cohausz teaches that the first programming module requests a set of features at p. 5, paragraph 2, sends a message to the programming module indicative of features, and the programming module returns a message; such that the programming modules interact with each other in response to user interaction with the first programming module, also at p. 5, paragraph 2.

Cohausz also teaches the following: that each of the plurality of display areas is individually and variable sized [claims 19, 22] at p. 5, paragraph 1, and p. 6, paragraph 2; the first window region always appears in front of application windows [claims 20, 23] at p. 4,

paragraph 5, wherein the status indicator is 'visible during the entire program'; and the first window region is in a 'private window layer' [claims 21, 24] also at p. 4, paragraph 5.

Claim Rejections - 35 USC § 103

7. The following is a quotation of 35 U.S.C. § 103 which forms the basis for all obviousness rejections set forth in this Office action:

> A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.
>
> Subject matter developed by another person, which qualifies as prior art only under subsection (f) or (g) of section 102 of this title, shall not preclude patentability under this section where the subject matter and the claimed invention were, at the time the invention was made, owned by the same person or subject to an obligation of assignment to the same person.

8. Claims 4 - 7 are rejected under 35 U.S.C. § 103 as being unpatentable over Cohausz (EPO 0 584 392 A1), based upon the English translation, and the patent to Mills et al. (U.S. patent 5,202,961).

Cohausz teaches that the display areas [individual fields 2] of the first window region [oblong field 1] are variably sized at p. 5, paragraph 1, and p. 6, paragraph 2, but does not teach that the first window region is variably sized [claim 4], such that none of the plurality of display areas is visible [claim 5], all are visible [claim 6], or a portion is visible [claim 7].

On the other hand, Mills et al., hereinafter Mills, teach that the size of the first window region is variable [claim 4] also at col. 4, lines 8 - 9, and also teaches sizing the first window region so that none of the display areas are visible [claim 5] with close box 28, at Fig. 2, and at col. 4, lines 7 - 8, or all [claim 6] or a portion [claim 7] of the display areas are visible, both at col. 4, lines 8 - 9.

Although Cohausz does not teach that the first window region is variably sized, it would have been obvious to one of ordinary skill in the art at the time of the invention to vary the size of the first window region as taught by Mills, because it gives the user control over how much and what to display of the status indicator.

Response to Amendment

9. Applicant's arguments with respect to claims 1 - 24 have been considered but are deemed to be moot in view of the new grounds of rejection.

The examiner agrees with applicant that Mills does not teach "providing logic that creates an operating environment like a shell for other programming modules to provide status and control functions". As pointed out by applicant, in Mills, the "control window is used for controlling video generated by an application".

Rather, the Cohausz reference was used to reject the present claims. Like applicant 's claimed invention, Cohausz teaches "creating an operating environment for a plurality of individual programming modules that provide status and control functions" to generate and

display a first window region [oblong field 1] having a plurality of display areas [individual fields 2], as shown at Figs. 1 - 3, and at p. 4, paragraph 5.

Conclusion

10. Applicant's amendment necessitated the new grounds of rejection. Accordingly, **THIS ACTION IS MADE FINAL**. See M.P.E.P. § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 C.F.R. § 1.136(a).

A SHORTENED STATUTORY PERIOD FOR RESPONSE TO THIS FINAL ACTION IS SET TO EXPIRE THREE MONTHS FROM THE DATE OF THIS ACTION. IN THE EVENT A FIRST RESPONSE IS FILED WITHIN TWO MONTHS OF THE MAILING DATE OF THIS FINAL ACTION AND THE ADVISORY ACTION IS NOT MAILED UNTIL AFTER THE END OF THE THREE-MONTH SHORTENED STATUTORY PERIOD, THEN THE SHORTENED STATUTORY PERIOD WILL EXPIRE ON THE DATE THE ADVISORY ACTION IS MAILED, AND ANY EXTENSION FEE PURSUANT TO 37 C.F.R. § 1.136(a) WILL BE CALCULATED FROM THE MAILING DATE OF THE ADVISORY ACTION. IN NO EVENT WILL THE STATUTORY PERIOD FOR RESPONSE EXPIRE LATER THAN SIX MONTHS FROM THE DATE OF THIS FINAL ACTION.

11. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Crescelle N. dela Torre whose telephone number is (703) 305-9782.

Any inquiry of a general nature or relating to the status of this application should be directed to the Group receptionist whose telephone number is (703) 305-3800.

Cnd
cnd
November 13, 1996

RAYMOND J. BAYERL
PRIMARY EXAMINER
ART UNIT 2415

CR/821(U)

Notice of References Cited

Application No.	Applicant(s)		
~~08/346,237~~	Christensen		
Examiner		Group Art Unit	Page 1 of 1
Crescelle Dela Torre		2773	

U.S. PATENT DOCUMENTS

	DOCUMENT NO.	DATE	NAME	CLASS	SUBCLASS
A					
B					
C					
D					
E					
F					
G					
H					
I					
J					
K					
L					
M					

FOREIGN PATENT DOCUMENTS

	DOCUMENT NO.	DATE	COUNTRY	NAME	CLASS	SUBCLASS
N						
O						
P						
Q						
R						
S						
T						

NON-PATENT DOCUMENTS

	DOCUMENT (Including Author, Title, Source, and Pertinent Pages)	DATE
U	EPO 0 584 392 A1, Cohausz, English Translation of the German patent document.	3/1992
V		
W		
X		

Translation of European Patent Application No. 584,392 A1
IPC: Cl^5: GO6F 3/033
Applicant and Inventor: Helge B. Cohausz
Application Date: August 28, 1992
Publication Date: March 2, 1994 (Patentblatt 94/09)
Original German Title: Statusanzeige

STATUS INDICATOR

Abstract

The invention relates to a status indicator for a computer program having an oblong field 1 on which an indicator field 4 or a cursor moves, whose respective position indicates the status on the oblong field 1, with the oblong field 1 comprising a plurality of individual fields 2 which are adjacent to one another. Each of these fields constitutes a control panel or control button, which, upon being activated, branches into the associated program area or executes the associated program function; and the indicator field 4 or the cursor is always located on the individual field 2 in whose associated program area/program function the user is currently located.

Specification

The invention relates to a status indicator in a computer program having an oblong field on which an indicator field or cursor moves, whose respective position indicates the status on the oblong field.

Different status indicators in the form of oblong fields are known. For example, oblong indicator instruments are known in which a cursor moves along a straight scale. It is also known how to arrange an oblong field at the edge of a window, especially of a text window, with a button field arranged at both ends of the oblong field in which an indicator button moves and is displaceable between both button fields, resulting in the content, especially the text, being moved around within the window. The position of the moving button on the oblong status field indicates only approximately at which location in the entire text the window is located, i.e., it must be estimated how far to move the button in order to reach a certain location in the text.

It is the objective of the invention to provide a status indicator which indicates very precisely at which location one is in the program, in a text, or in an information range, with only a simple operation being required.

The problem under consideration is solved according to the invention in that the oblong field comprises a plurality of individual fields which are adjacent to one another, each of

which constitutes an operating field or a control button which, upon being activated, branches into the associated program area or executes the associated program function, with the indicator field or the cursor always being located on the individual field in whose associated program area/program function the user is currently located.

With this type of status indicator the indicator field or cursor moves over the individual fields and thus indicates very precisely at which location in the program or information one is located, since the individual fields represent portions of the individual program, text or information, i.e., sections, paragraphs, chapters or segments of information. The respective location of the program, text, or information segment is thus precisely indicated. Moreover, the individual fields are control panels or control buttons, which, when activated (clicked on) lead to the respective program area, text, or information segment. The status indicator thus has the double function of operating like a menu and of displaying exactly where in the program or in the body of information the operator or user is located.

It is especially advantageous that the indicator field or the cursor assumes a plurality of positions in or on the individual fields, depending on where the user is in the sub-area or the associated program area. This allows for an even finer division of the individual fields, i.e., the indicator field, or cursor can be located at different locations within the

individual field and can therefore show in which area of the associated program area the user is.

It is especially advantageous if the indicator field or the cursor is colored and/or transparent. It is also especially advantageous if the size of the indicator field or cursor is equal to or smaller than the individual field.

A very precise indication is provided if the width of the indicator field or cursor is a fraction of the width of an individual field, with the fraction corresponding to the number of sub-areas of the program areas.

Embodiments of the status indicator are shown in Figs. 1 to 3.

Fig. 1 is a status indicator in the form of an oblong field 1 comprising a plurality of individual fields 2, with the individual fields, which are of equal height but differ in length, being arranged closely adjacent to one another. This type of oblong field 1 is already marked in the embodiment and it is usually arranged horizontally at the upper and lower edge on the monitor screen where it is visible during the entire program. As an alternative or additionally, the oblong field can also be arranged vertically at the left and/or right edge of the screen. Two or more oblong fields may also be arranged horizontally, vertically, or parallel to one another. Each individual field is associated with a defined program area or a defined program function, with the individual fields being arranged successively in accordance with the logical and/or timed running of the

program. In this case, important and/or large program or content areas can be represented by a longer and/or larger individual field than smaller or less important program areas or program functions.

One of the individual fields is activated by clicking a mouse cursor or typing with a finger such that the activation of the individual field results in a function corresponding to the respective program area or respective program function, i.e., switching to a program area or a program function occurs. In this case the oblong status indicator can represent a menu in which the individual fields represent menu points or menu subjects.

Along the oblong status indicator and therefore, field 1, an indicator field or cursor 4 moves which indicates at which location in the program or in which program function the user of the program is (this user may also be a machine, a computer or a program). In the embodiment according to Fig. 1 a two-part indicator field 4 is used which moves with an upper region along the upper edge and with a lower region along the lower edge of the oblong field 1. The indicator field or the cursor 4 can always have the same length as an individual field 2, such that the indicator field/cursor 4 skip from one individual field to another; see the embodiment according to Fig. 3, outer right. right.

Alternatively, the indicator field/cursor 4 may also be shorter than an individual field 2 such that the position of the