**PCT** WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

| (51) International Patent Classification 5 : | | (11) International Publication Number: WO 93/03473 |
|---|---|---|
| G09G 1/16 | A1 | (43) International Publication Date: 18 February 1993 (18.02.93) |

(21) International Application Number: PCT/US92/06634

(22) International Filing Date: 6 August 1992 (06.08.92)

(30) Priority data:

| | | |
|---|---|---|
| 741,103 | 6 August 1991 (06.08.91) | US |
| 839,901 | 21 February 1992 (21.02.92) | US |

(71) Applicant: BEAGLE BROS, INC. [US/US]; 6215 Ferris Square, Suite 100, San Diego, CA 92121 (US).

(72) Inventors: STAW, Michael ; 85 Brainerd Road, Boston, MA 02134 (US). SIMONSEN, Mark, S. ; 12716 Fairbrook Road, San Diego, CA 92131 (US). HOULE, Michael ; 200 Parkview Street, Manchester, NH 03103 (US). FRANK, Richard ; 68 Brookside Avenue, Newtonville, MA 02160 (US). ROSENBLUM, Bruce ; 33 Fairway Drive, West Newton, MA 02165 (US). TEPPER, Kenneth, A. ; 101 Greenlawn Avenue, Newton, MA 02159 (US). SISAK, Steve ; 87 Bristle Street, Apartment 3A, Cambridge, MA 02139 (US).

(74) Agents: SIMPSON, Andrew, H. et al.; Knobbe, Martens, Olson & Bear, 620 Newport Center Drive, Suite 1600, Newport Beach, CA 92660 (US).

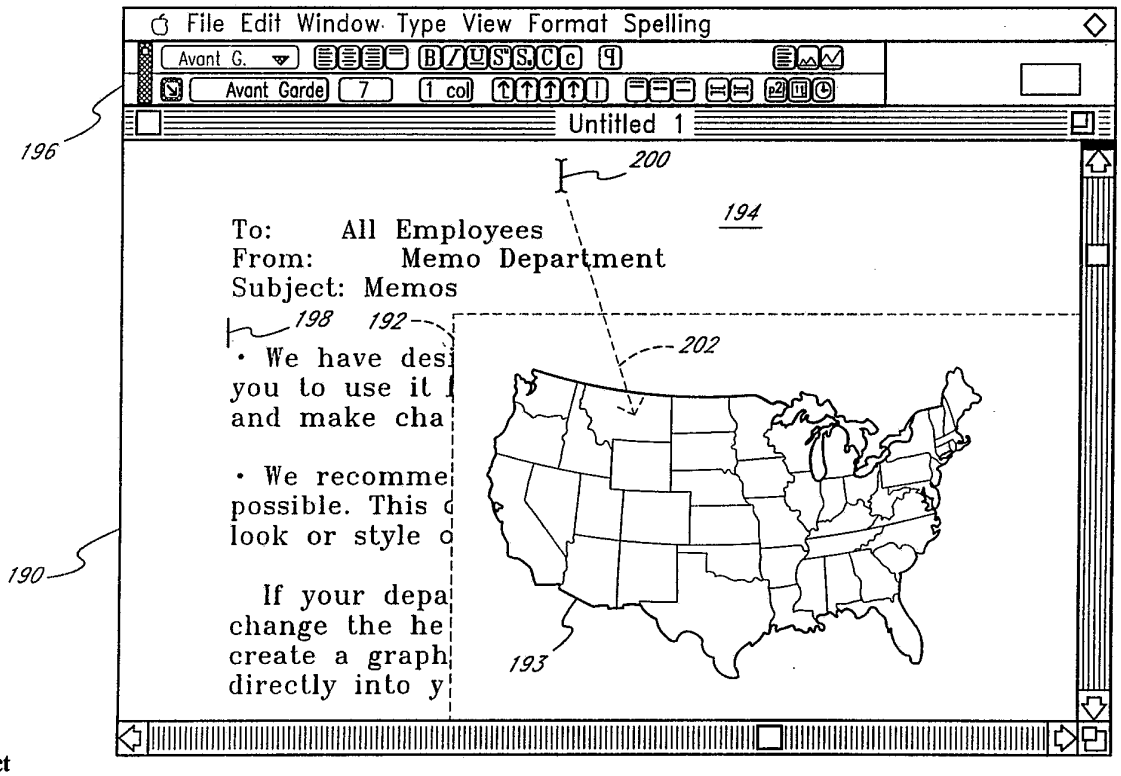
(81) Designated States: AU, BB, BG, BR, CA, CS, FI, HU, JP, KP, KR, LK, MG, MN, MW, NO, PL, RO, RU, SD, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, SN, TD, TG).

**Published**

*With international search report.*

*Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.*

(54) Title: IN-CONTEXT EDITING SYSTEM WITH FRAME TOOL




(57) Abstract

An in-context editor comprising a kernel (214) and one or more modules (184). The in-context editor includes a frame tool for dynamically creating, positioning and object type selecting a document (194) while another document (192) is simultaneously displayed.

*FOR THE PURPOSES OF INFORMATION ONLY*

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

| | | | | | |
|---|---|---|---|---|---|
| AT | Austria | FI | Finland | MN | Mongolia |
| AU | Australia | FR | France | MR | Mauritania |
| BB | Barbados | GA | Gabon | MW | Malawi |
| BE | Belgium | GB | United Kingdom | NL | Netherlands |
| BF | Burkina Faso | GN | Guinea | NO | Norway |
| BG | Bulgaria | GR | Greece | NZ | New Zealand |
| BJ | Benin | HU | Hungary | PL | Poland |
| BR | Brazil | IE | Ireland | PT | Portugal |
| CA | Canada | IT | Italy | RO | Romania |
| CF | Central African Republic | JP | Japan | RU | Russian Federation |
| CG | Congo | KP | Democratic People's Republic of Korea | SD | Sudan |
| CH | Switzerland | KR | Republic of Korea | SE | Sweden |
| CI | Côte d'Ivoire | LI | Liechtenstein | SK | Slovak Republic |
| CM | Cameroon | LK | Sri Lanka | SN | Senegal |
| CS | Czechoslovakia | LU | Luxembourg | SU | Soviet Union |
| CZ | Czech Republic | MC | Monaco | TD | Chad |
| DE | Germany | MG | Madagascar | TG | Togo |
| DK | Denmark | ML | Mali | UA | Ukraine |
| ES | Spain | | | US | United States of America |

# IN-CONTEXT EDITING SYSTEM WITH FRAME TOOL

## Background of the Invention

### Field of the Invention

The present invention relates to the editing of multiple documents and, more particularly, to editing a composite document having one or more embedded documents.

### Description of the Prior Art

In most general purpose computer systems, the computer user is provided a mechanism whereby text documents can be created and modified at will. The mechanism for such text editing is most commonly referred to as a word processor which by this definition is a type of computer program. Many modern computers, including microcomputers that are today commonly used in the home and office, also contain mechanisms for creating and editing graphics, numerics and other types of objects. As examples of other types of editors, certain types of graphic objects can be edited with line drawing or "draw" programs and other graphic objects require less sophisticated editing features which are satisfied by so-called "paint" programs, and some numeric objects can be edited with spreadsheet programs.

At some point in the development of these various editors it was realized that varying types of objects could be combined in a single document providing information in the format that is usually associated with professional publications such as magazines and books. For instance, where illustrations would enhance a written report a desktop publishing system may be utilized for formatting graphics and text in the document. Some technologists have even developed integrated editors that operate on different types of objects in a document which is stored in a computer in only a single file. Such a system was disclosed by Barker, et al. (U.S. Patent No. 4,815,029).

However, it has been observed that rather than operating on the contents of documents, as with integrated editors, greater benefits are realized by operating on the containers

of the contents, or the documents themselves. Principally, it is desirable to be able to embed documents in other documents so that the contents of documents can be mixed and matched, and re-used in an environment that fosters creativity. For example, in an embedded document environment, once a library of standard paint objects including maps, logos, digitized images, and the like, is created, it becomes relatively straightforward for a user to pick an existing picture and "plug" or embed the picture into a text document.

Taking the notion one step further, if a document that is embedded by an embedding document changes, it is often desired that the change be reflected in the embedding document as well. This feature has been previously implemented in the Jazz program, licensed by Lotus Development Corp. of Cambridge, Massachusetts, wherein the embedding is termed HotView. Jazz allows spreadsheet, graph, database and word processing data to be embedded in a word processing document. More recently, the Microsoft Corporation has introduced data linking using dynamic data exchange (DDE), a data interchange protocol that permits Windows 3.X applications to share data, and its enhancement called object linking and embedding (OLE). However, none of these or other present systems allow a user to treat the data of a document within a document and the surrounding document's data as a single entity that may be manipulated in a seamless environment, i.e., viewing and modifying both documents in a single context such as a window.

Other software products, recently introduced to the marketplace, have attempted to combine editing functions for multiple objects by also allowing the user to create new objects in a selected region of a document. For instance, AmiPro, Version 1.21, licensed by Samna, permits a user to define a region of the document, select an object type and create new objects of the selected type in the defined region. However, AmiPro does not create a separate window for the defined region and, moreover, saves the objects of the defined region and the objects outside of the region, together in a single document.

One spreadsheet program EXCEL 3.0, licensed by Microsoft Corporation, provides the user with the capability to create a histogram document in a window while the underlying spreadsheet document is displayed in another window. Nonetheless, the histogram is not related to, or displayed at, any particular location in the spreadsheet. Also, while a change in the spreadsheet data will be reflected in the histogram, there is no mechanism to edit the histogram independently of the spreadsheet. Furthermore, if the histogram is "pasted" into another document, changes in the spreadsheet data will not affect the pasted histogram.

Consequently, a need exists for a system that provides for creating and modifying embedded documents while an embedding document is simultaneously displayed.

## Summary of the Invention

The present invention satisfies the aforementioned needs by providing an in-context editing system with a frame tool. The invention allows users to mix documents of selected object types and place a document or document portion in any location(s) in another document to form a composite document. The user of the in-context editor (ICE) can edit the contents of an embedded or subscriber document without leaving the context of the subscribing document in which the subscriber document is embedded. In addition, when a subscribing document, displayed in a parent window, is scrolled, the frame scrolls with the parent.

The present invention provides, in a computer, a method of editing a plurality of documents, comprising the steps of:displaying at least a portion of a selected first one of the documents; linking a selected second one of the documents to the first document; displaying at least a portion of the second document simultaneously with said displaying of the first document; positioning the second document at a selected location in the first document so that together the

selected documents are displayed as a composite document; and modifying the second document while the composite document is displayed.

One aspect of the present invention provides, in a computer, a method of creating a composite document, comprising the steps of: displaying at least a portion of a first document containing an object, selecting a frame in the displayed first document portion, selecting an object type from a plurality of object types, creating a second document for containing an object of the selected object type which is displayable in the frame so that together the documents are displayed as a composite document, linking the second document to the first document, and creating an object of the selected object type in the frame while the composite document is displayed. Furthermore, the method provides for modifying an object in the second document.

These and other objects and features of the present invention will become more fully apparent from the following description and appended claims taken in conjunction with the accompanying drawings.

## Brief Description of the Drawings

Figure 1 is a functional block diagram of a representative computer used by one preferred embodiment of the present invention;

Figure 2 is a screen display of "windows" that may be produced on the video display of the computer shown in Figure 1;

Figure 3 is an exemplary diagrammatic view of document linking as constructed by the present invention;

Figure 4 is a detailed diagram of a preferred document linkage constructed by the present invention;

Figure 5 is a screen display of a composite document formed from a subscriber document and an intermediate document, the subscriber and intermediate documents having nonhomogeneous objects (i.e., text and graphics), as provided by a preferred embodiment of the present invention;

Figure 6 is a hierarchical chart of the preferred class architecture in one preferred embodiment of the present invention;

Figure 7 is a flow diagram of the "TLocalKernel::EventLoop" function of the presently preferred embodiment of the in-context editor (ICE) of the invention;

Figure 8 is a flow diagram of the "TLocalKernel::DispatchOSEvent" function used by the in-context editing function of Figure 7;

Figure 9 is a flow diagram of the "TLocalKernel::DoMouseDown" function used by the in-context editing function of Figure 8;

Figure 10 is a flow diagram of the "TDocumentWindow::DoMouseDown" function used by the in-context editing function of Figure 9;

Figure 11 is a flow diagram of the "TFrame::DoMouseDown" function used by the in-context editing function of Figure 10;

Figure 12 is a screen display of the same composite document as illustrated in Figure 5, which is produced after activating and editing the intermediate document using the presently preferred in-context editor (ICE) of the invention;

Figure 13 is a flow diagram of the "TSubscriptionList::HitTestFloaters" function used by the in-context editing function of Figure 11;

Figure 14 is a flow diagram of the "TSubscriptionList::LaunchPublisherFromToken" function used by the in-context editing function of Figure 11;

Figure 15 is a flow diagram of the "TSubscriptionList::DrawFloaters" function used by the presently preferred embodiment of the in-context editor (ICE) of the invention;

Figure 16 is a screen display of the same composite document as illustrated in Figure 12, after the composite document is scrolled by the presently preferred embodiment of the in-context editor (ICE) of the invention;

Figure 17 is a flow diagram of the "TDocument::CloseLink" function used by the in-context editing function of Figure 10; and

Figure 18 is a screen display of the same composite document as illustrated in Figure 12, after the window containing the intermediate document is made invalid.

Figure 19 is a screen display of a portion of a first document having a frame selected using the presently preferred in-context editor (ICE) of the invention;

Figure 20 is a screen display of the first document and frame, of Figure 19, with a list of various object types presented for selection;

Figure 21 is a screen display of the first document and frame, of Figure 19, after an object type has been selected and a second document has been created;

Figure 22 is a flow diagram of the "TToolbar::DoMouseDown" function used by the in-context editing function of Figure 9;

Figure 23 is a flow diagram of the "TLocalKernel::HandleCreateFrame" function used by the in-context editing function of Figure 22;

Figure 24 is a flow diagram of an extended portion of the "TFrame::DoMouseDown" function, shown in Figure 11, which relates to the frame mode of the present invention;

Figure 25 is a flow diagram of the "TLocalKernel::CreateDocInFrame" function used by the portion of the in-context editing function of Figure 24;

Figure 26 is a screen display of the first document and frame, of Figure 19, wherein the frame includes the second document of the selected object type having draw objects;

Figure 27 is a screen display of the composite document of Figure 26 wherein the in-context editor is no longer in frame mode;

Figure 28 is a screen display of the composite document of Figure 27 after modification of the second document as simultaneously displayed in the child window; and

Figure 29 is a screen display of the composite document

of Figure 28 showing the composite document of first and second documents displayed in a single window.

Detailed Description of the Preferred Embodiments

Reference is now made to the drawings wherein like numerals refer to like parts throughout.

Figure 1 illustrates the functional components of a computer, of the presently preferred invention, generally indicated at 100. The functional block diagram represents a Macintosh SE/30 manufactured by Apple Computer of Cupertino, California. Although the preferred computer 100 may be referred to herein as a Macintosh, it will be understood by computer technologists that many other computers may be used in place of the one described.

The preferred computer 100 comprises a microprocessor 102. By way of example, in the Macintosh computer, the microprocessor 102 is one of the Motorola 680x0 family, such as the 68030 in the Macintosh SE/30. The microprocessor 102 executes the instructions of computer programs (not shown) stored in a read-only memory (ROM) 104 and/or a random-access memory (RAM) 106. The instructions are communicated to the microprocessor 102 via an address and data bus 108 which is 32 bits wide for both address and data. The ROM 104, in the preferred computer 100, comprising 256 kilobytes of memory, stores the Macintosh Operating System, or "OS". The on-board RAM 106 is expandable from 1 Megabyte up to 128 Megabytes and stores application programs and data which are included by the in-context editor (ICE) of the present invention. A coprocessor 110, such as, for example, the Motorola 68882 floating-point unit, advantageously improves the performance of applications which use many floating-point operations such as graphics processing that may be included in the present invention.

The ICE of the present invention can be magnetically encoded on a portable media such as a floppy disk (not shown) which may be placed in a floppy disk drive 111. The floppy disk drive 111 is controlled by a floppy disk drive controller

112 which principally communicates with the components 102, 106 across the buses 108. In this way, the ICE can be selectively loaded into the RAM 106 and executed by the microprocessor 102. For more convenient and faster storage, the ICE can be transferred and stored on a hard disk drive 114. In the preferred computer 100, the hard disk drive 114 is interfaced to other computer components, principally the components 102, 106, by way of a Small Computer System Interface (SCSI) controller 116, which is an industry standard interface that provides high-speed access to peripheral devices.

A user (not shown) most often provides input data to the computer 100 by moving and "clicking" a mouse, or other pointing device 118 and typing on a keyboard 120. The input devices 118, 120 communicate with the computer 110 through an input device controller 122. In the preferred "what-you-see-is-what-you-get" (WYSIWYG) environment of the Macintosh, the mouse 118 guides a pointer (indicated in Figure 2 by the arrow 133) across a video display 124. The video display 124 is refreshed by the computer 100, via a video controller 126, so as to apprise the user of the current state of processing.

Figure 2 illustrates a representative screen display that may be produced on the video display 124 of the computer 100 (Figure 1) using either the present technology or the in-context editing system of the present invention. The preferred embodiment of the invention includes a desktop, or "windows", interface that is a metaphor for working with documents on a desktop. One such interface is provided for PC/AT, or Industry Standard Architecture (ISA) bus, computers by the Windows program licensed by Microsoft Corporation of Redmond, Washington. However, the preferred desktop interface, used by the existing Macintosh software and generally also made available by the present invention, is more fully described in Human Interface Guidelines: The Apple Desktop Interface, Apple Computer Inc., Addison-Wesley Publishing Co., Inc., 1987, which is hereby incorporated by reference herein. To more completely comprehend the present

invention, and appreciate the advantages thereof, certain portions of the desktop interface will be described with respect to Figure 2.

The screen display of Figure 2 includes a menu bar 130, which is a horizontal strip at the top of the screen, that includes one or more menu titles such as the title "File" indicated at 132. A pointer 133 can be selectively moved across the screen (via movement of the mouse 118 of Figure 1) and placed on top of a menu title. The user may then depress a button on the mouse 118, called a "click", and a pull-down menu such as the one at 134 is displayed to the user. (As an alternative to the mouse 118, the user may move and "click" at the location on the screen indicated by the pointer 133 using the keyboard 120, a track ball (not shown) or some other input device.) The menu 134 includes various operations (or verbs) that may be carried out on a preselected object (or noun).

The majority of the screen, underneath the menu bar 130, is dedicated to a desktop 136, a portion of which is illustrated in Figure 2. In general, if the user clicks the mouse 118 on the desktop 136, there is no change in the screen display. The desktop 136 includes icons, which are small pictures that represent objects to be worked on, such as the trash can icon 138. Although not shown, by "double clicking" the mouse 118 (i.e., two successive, closely spaced in time, depressions of the mouse button), disk icons can be opened as windows into documents which appear to rest on the desktop 136.

The screen display of Figure 2 illustrates two overlapping windows "on" the desktop 136. An inactive window 140, wherein the inactivity is indicated to the user by a lack of shading at the edges of the window including a lack of "racing stripes" in the top edge of the window, has a content area 141 that includes textual data or objects. The inactive window 140 underlies an active window 142 which has a content area 143 showing graphical data or objects. In the desktop metaphor, only one window is active at a time (always at the top of, or in front of, any overlapping windows) and the

active window is allowed to have operations performed on its objects. The area of window surrounding the contents is sometimes referred to as the window frame, although frame may also refer to the entire window.

It should be noted that in the usual desktop interface, the applications, e.g., word processor and paint programs, have completely independent windows that have no "knowledge" of one another.

The structural elements of the window frame, which surround the content area of a window, are highlighted in the active window 142. Most of the structural elements can be selected by clicking the mouse 118 (Figure 1) or using the mouse 118 to "drag" the element across a portion of the video display 124. The mouse 118 can be used to drag a displayed element by positioning the pointer 133 on the element, depressing the mouse button, and moving the mouse 118 in the desired direction while holding the button down. Briefly, the structural elements are as follows: a close box 144 (upper left corner of window, in title bar 146), which when clicked closes the window so that whatever previously underlaid the window is now displayed; a title bar 146 (horizontal strip at the top of the window) for indicating the document being displayed in the window and for dragging the window; a zoom box 148 (upper right corner of window, in title bar 146) which when clicked makes the window enlarge or reduce to one of two preselected sizes; a vertical scroll bar 150 (vertical strip on the right side of the window) which represents the vertical dimension of the entire document; a size box 152 (lower right corner of the window) which when clicked allows the user to "grow" the window or change its size in selected increments; a horizontal scroll bar 154 (horizontal strip at the bottom of the window) which represents the horizontal dimension of the entire document; a scroll box 156 (one in each scroll bar 150, 154) which when dragged within the scroll bar "moves" the document in the dragging direction; and a scroll arrow 158 (at the ends of the scroll bars 150, 154) which when clicked "moves" the document a distance of one unit in the chosen

direction, e.g., one line for a word processing program, one row or column for a spreadsheet program, and so forth.

Now referring to Figure 3, the complete independence of documents in the desktop interface, noted previously hereinabove, can be overcome by a procedure known as "warm-linking" or "hot-linking". In the presently preferred embodiment of the in-context editing invention, such linking of documents is achieved by taking advantage of a concept, introduced by Apple Computer in System 7, known as "publish-and-subscribe" (more fully described in Macintosh Reference, System 7, Apple Computer, Inc., 1991 which is hereby incorporated by reference herein). Using publish-and-subscribe, material from a publishing document, called a publisher, can be copied and pasted to one or more subscribing documents (the material is then termed a subscriber) such that when the original material is updated in the publishing document, the changes are automatically reflected in the subscribing documents.

By way of example, in Figure 3, a publishing document 162, preferably stored on the hard disk 114 (Figure 1)), contains a set of bar chart objects. The user selects a publisher 164 by outlining certain bar chart material to be published from the document 162 using the mouse 118 and the video display 124. The selection function is accomplished in a window such as the active window 142 shown in Figure 2. Thereafter, the publisher 164 is saved in an edition file 166 on the hard disk 114.

At some time after the edition file 166 has been created, the user opens a window for a first subscribing document 168 that is preferably stored on the hard disk 114. The window is opened by clicking the "File" title in the menu bar 130, and then selecting "New" for a new document or choosing an existing document that is typically stored on the hard disk 114 (Figure 1). The subscribing document 168 is, in the example, a text document that is manipulated, or edited, with a word processing program. The user employs the mouse 118 to create and position a subscriber 169 in the document 168. The

subscribing document 168, now containing text, and a link to the bar chart objects saved in the edition file 166, is saved back to the disk 114. A similar procedure is followed for a second subscribing document 170 and an associated subscriber 171. Both of the subscribers 169, 171 are now linked to the edition file 166 which in turn is linked to the publishing document 162.

If the publishing document 162 is now opened in a window, and the bar chart objects in the publisher 164 are modified by a bar chart editor (not shown), the modifications are saved in the publishing document 162, as well as the edition file 166. When the subscribing documents 168, 170 are next printed, displayed, etc., the subscribers 169, 171 will contain the modifications made previously in the publishing document.

The presently preferred in-context editor (ICE) of the invention employs a linking concept that in some ways is similar to the publish-and-subscribe concept just discussed, and that is compatible with Apple's System 6 and System 7 software. For example, in the present invention, multiple subscribing documents may subscribe to the same subscriber. However, the present linkage mechanism extends the concept by including some amount of redundancy which aids in restoring the links after one or more files/documents have been lost or corrupted. As shown in Figure 4, a publishing document 174 contains a publisher 176. The user-definition of the publisher 176 causes an intermediate file 178 (which is to be distinguished from an edition file) to come into existence. The intermediate file 178 contains a segment 179 of the published document 174, a publisher record 180 which refers to the location of the segment 179 in the publishing document 174, a file specification (FSSPEC) record 181 (used by System 6 software) identifying the publishing document 174, an alias record 182 (used by System 7 Software) identifying the publishing document 174 more robustly than by file specification, a preview 183 that is a miniature picture of the published segment 179 (for display to the user by the "GetInfo" command in the "File" menu or the "Subscribe to..."

command in the "Edit" menu) and a module descripter 184 that identifies the module that created the segment 179, e.g., a specific word processing program. Information as to the module that created the segment 179 is important for the later editing of the segment 179 in the context of a subscribing document 186 as carried out by the in-context editing of the present invention. A subscribing document 186 subscribes to a subscriber document 188, or in different terms, an embedding document 186 includes an embedded document 188.

Figure 5 illustrates a screen display that is produced on the video display 124 (Figure 1) by the preferred in-context editor (ICE) of the present invention. The ICE includes a set of modules, presently including: word processor, database, spreadsheet, chart, draw, paint and communications. The ICE modules may communicate to the desktop interface via a kernel program which will be described below. Since the general functions of the ICE modules are well understand in the present technology the specifics of each will not be discussed except to say that each module interacts with a specific window and document via the kernel.

The ICE display looks similar to the desktop interface display previously discussed with respect to Figure 2. However, in this display, a parent window 190 shows that two documents, a subscriber document 192 comprising a paint object 193 created with a paint program (not shown), and a subscribing document 194 having text objects, have been integrated into a single, composite document (but each member document of the composite document is stored at a different logical location of the disk 114 (Figure 1). A tool bar 196, showing operations for text objects, reflects the fact that text editing may at the time be accomplished in the context of the subscribing document 194. For example, text may be inserted at the insertion point 198 when the user types on the keyboard 120. A similar display could be generated with the publish-and-subscribe feature of the prior technology. However, the prior technology would <u>not</u> allow editing the object in the subscriber document 192 in the context of the

subscribing document 194, or in other words, editing the embedded document without leaving a view into the embedding document. In the present invention, the user may initiate in-context editing, (or editing in place without leaving a view of the composite document), by first moving the pointer 200 as indicated by the arrow 202 inside of the subscriber document 192, and then double clicking the button on the mouse 118.

Figure 6 illustrates the presently preferred class architecture of the in-context editor. A "class" is a definition of a data type that specifies the representation of objects of the class and the set of operations that can be applied to the objects. An object of a class is a region of storage or memory. The notion of a "class" will be understood by those skilled in the object-oriented programming technology and, in particular, by those familiar with the "C++" programming language. Classes may be more fully comprehended by reference to The Annotated C++ Reference Manual, Ellis, Margaret and Stroustrup, Bjarne, Addison-Wesley Publishing Co., 1990. The rationale for programmer-defined classes is that it provides data abstraction by allowing the representation details of objects to be hidden and accessed exclusively through a set of operations defined by the class. For example, putting a class in human terms, a "bakery class" would provide a mulberry pie sale operation to allow a customer to purchase a mulberry pie without any knowledge on the part of the customer as to how pies were stored at the bakery, or how the customer's pie was selected from a number of different pies.

In Figure 6, the superclass of all classes, i.e., the class from which all other classes are derived and inherit their properties, is a TObject class 206. TObject 206 provides the methods for objects to be created, the strategy for dynamic memory allocation and deallocation, and the method for receiving events and passing events between objects. An event is a significant, asynchronous change in the system that requires processing. In this sense, the present invention is "driven" by events since nothing happens in the absence of an

event. An example of an event is a click of the mouse 118 (Figure 1), i.e., the depression of the button.

A TEventHandler class 207 is derived from TObject 206. TEventHandler 207 provides an object which can react to an event that is sent by another object. TEventHandler 207 thus provides a common event passing interface between its subclasses. A TPublicationList class 208 and a TSubscriptionList class 209 are also derived from TObject 206. TPublicationList 208 is a class that manages the links to document sections, or publishers, that have been published by a publishing document. TSubscriptionList 209 is a class that manages the subscribers, or embedded documents, of a subscribing document.

The remaining classes are subclasses of TEventHandler 207. The TDocument 210 and TModule 212 classes (both of which have an asterisk "*" to indicate that they are external access points into the kernel) are classes from which authors of ICE modules may derive classes to send and receive events to objects not in their own class. The TKernel 214, TMenu 216 and TMenuBar 218 classes handle the desktop interface (as described above) except for windows. A TVisibleThing class 220 maintains the entire visual hierarchy on the desktop that defines windows.

A TWindow class 222 is a subclass of TVisibleThing 220. TWindow 222 handles events that relate to a window opened by a dynamically allocated copy of a TWindow class. TDocumentWindow 224 is a subclass of TWindow 222. Since the ICE allows multiple documents to be associated with a parent window, such as 190 in Figure 5, TDocumentWindow provides the mechanism to dispatch an event through the pane (the contents of a window may be viewed in up to four different panes but typically there is a one-to-one correspondence between the content area and a single pane) where the subscriber is located.

TPane 226 is a subclass of TVisibleThing 220. TPane 226 handles events that are directed to a pane of a window. TFrame 228 is a container class for up to four panes.

TDocumentView 230 is a subclass of TPane 226 that links a document to a visible area. TVirtualPane 232 is a subclass of TDocumentView 230 that does coordinate system mapping for modules that require such mapping, presently the paint and draw modules.

Software for the in-context editor (ICE) is preferably written using the Macintosh Programmers Workshop (Version 3.2) licensed by Apple Computer. The software described herein was translated from source code to machine-readable object code using the "C++", "C" and Pascal language compilers and utilities, as well as an assembler for the 680x0 family of microprocessors, all of which are computer programs in the Workshop. Nonetheless, one skilled in the technology will recognize that the steps in the accompanying flow diagrams can be implemented by using a number of different compilers and/or programming languages.

It should also be observed that the following flow diagrams are only meant to represent the functional equivalents of their named source code counterparts, and so, the diagrams may include material that does not completely parallel the named location of the function.

As was mentioned above, the ICE modules, e.g., word processor, each of which may be associated with a different type of document, communicate with the desktop interface by way of a kernel, or ICE kernel, which is a portion of the ICE software. Hence, other than the document that is currently active, each module has no knowledge of other documents, which are subscribers belonging to a single subscribing, or parent, document. The kernel maintains the knowledge or states of in-context editing by handling events that relate to the standard desktop interface (e.g., scrolling) and feeding module specific events, typically operations in the content area of a window, directly to the creator module of the specified document.

The top-level control flow of the ICE kernel which is the primary interface between modules, documents and windows is illustrated in Figure 7 as an event loop function. The event

loop of the preferred ICE is entitled TLocalKernel::EventLoop (the double colon "::" indicates ownership of the named function following, by the named class preceding). TLocalKernel is a subclass of the TKernel class 214 shown in Figure 6.

An event loop function is a standard type of function that is written by Macintosh applications programs to receive events in the event queue from the OS. The particular event loop function of the ICE kernel is entered at a start state 236 and proceeds to a decision state 238 to test whether the program has been terminated by the user clicking on the "Quit" operation of the "File" menu. If the program is not done, the kernel moves to a state 240 to wait on the next event by making a call to the Macintosh Operating System (OS). The next event from the event queue, a time and priority ordered sequence of events, could be a "mouse down", i.e., the button of the mouse 118 (Figure 1) is depressed, "mouse up", i.e., the mouse button is in its normal, or released, position, "key down", i.e., a key on the keyboard 120 is depressed, and so on. Once an event is received from the OS, which could occur at any time, the kernel proceeds to a function 242 to dispatch the event and then returns to the top of the event loop, state 238 to continue processing events. Assuming that the user quits the program, the event loop moves from state 238 to an end state 244 and returns control to the OS.

The dispatch event function 242 of Figure 7 is preferably implemented by TLocalKernel::DispatchOSEvent which is illustrated diagrammatically in Figure 8. After the function is preferably entered at a start state 248, the kernel proceeds to a decision state 250 to test whether the event is a key down, key up or idle event. Idle events are returned by the OS when a request to get an event from the event queue is made and the queue is empty, i.e., no events are pending.

If the event is not one of those enumerated, the kernel proceeds to a state 252 wherein it closes the visible regions of child windows thus hiding the child windows from the OS. A child document, or subscriber document, is a document that

is embedded in the parent document, or subscribing document. A child document may be modified by activating a child window which will be further discussed below. The visible regions of child windows must be closed to the OS so that the kernel can intercept, or trap, all clicks in the parent window including those located in child documents.

From either the state 252, or the unsatisfied condition of state 250, the kernel moves to a state 254 wherein the event is processed according to its type. Of the greatest significance for further discussion is the handling of a mouse down (click) event.

Figure 9 illustrates the flow diagram for the mouse down event function, called "TLocalKernel::DoMouseDown", which performs part of the "do" event state 254 of Figure 8. The kernel enters the mouse down function at a start state 258 and then moves to state 260 where it queries whether the clicked pointer 133 (Figure 2) is in the menu bar 130. If the test is successful, the kernel proceeds to state 262 wherein it selects the module function of the active window (the module function associated with the clicked menu title). A test is then made at decision state 264 to determine whether a window handler has been requested due to a mouse down event being received in some portion of a window. If no window handler was selected, as was the case of a menu selection, the kernel terminates the function at an end state 266.

Now, if the test at the state 260 was unsuccessful, the kernel moves to a decision state 268 and tests whether the click (i.e., the pointer 133 (Figure 2) was at a particular location on the screen display provided by the video display 124 (Figure 1) and the mouse button was depressed) was in the system window (not shown) as defined in older Macintosh systems. If the click is in the system window, the system click is handled in state 270, and the kernel terminates the function as previously described through states 264 and 266.

If the test at the state 268 was unsuccessful, the kernel moves to a decision state 272 and tests whether the click was in one of the boxes (i.e., 144, 148, 152, 156 and 158 of

Figure 2) and, if so, creates a mouse track event in state 274. The mouse track event is created as a record of mouse position for drag processing. The window handler associated with the "hit" window (i.e., the window where the click was located) is identified at state 276, and after succeeding at the test in state 264, the kernel moves to state 277 and sends the event to the window handler following which the kernel terminates the function at state 266.

If the test at the state 272 was unsuccessful, the kernel moves to a decision state 278 and tests whether the click was in the content area of the window (e.g., the composition of document areas 192 and 194 in the window 190 of Figure 5). If the click was in the content area, the window handler associated with the "hit" window is identified at state 280 and the kernel proceeds as described above through the states 264, 277 and 266.

If the test at the state 278 was unsuccessful, the kernel moves to a decision state 282 and tests whether the click was in the desktop (e.g., 138 of Figure 2). If the test is successful, a mouse down event is created at state 284 and the kernel terminates the function through the states 264 and 266 as previously discussed. Otherwise, if the click is not in the desktop (the test at state 282), the click is not handled and the kernel terminates the function through the states 264 and 266.

Figure 10 illustrates the window handler function TDocumentWindow::DoMouseDown which receives the mouse down event from TLocalKernal::DoMouseDown at state 277, Figure 9. After the function is entered at the start state 286, the kernel moves to a decision state 288 and tests whether the window is active (see, e.g., the inactive and active windows 140, 142 of Figure 2). If the window is not active, the window is made active at a state 290 by bringing the window to the top, or front, of any other windows and "turning on" the window frame, and then the kernel exits the function at an end state 292.

On the other hand, if the window was found to be inactive

at state 288, the kernel moves to state 294 and finds the portion of the window (e.g., frame or content portion) that was clicked. If it is determined at decision state 296 that an active (or "live") child document does not exist (the case of Figure 5, i.e., there is no window drawn around the child document, or subscriber document), the clicked portion of the window is handled at a state 298. The function accomplished in state 298 is more fully described hereafter with respect to Figure 11. From the state 298 the kernel moves to state 292 wherein the function terminates.

If, at the decision state 296, it is determined that the window does contain an active child document, the kernel proceeds to a decision state 300 to determine whether the click was in the active child document. If the click was not in the active child, the kernel moves to a state 302 wherein the parent document, or subscribing document, is identified as the document that is the focus for future clicks. Thereafter, at state 304, the active child's window is closed (i.e., the window frame around the child document will not be displayed at the next refresh of the video display 124 (Figure 1)) and the parent document is marked active by the kernel so that subsequent user events (e.g., operations such as "Cut" and "Paste") may be directed to the parent document. Continuing to the end state 292, the kernel terminates the function.

Returning to decision state 300, if the click was in the active child document, the kernel moves to a state 306 wherein the visible region of the child is restricted to be only that portion of the child window (also referred to as a "floater" since child windows do not have shadowing as in the standard Macintosh window and hence they appear to "float" on the "surface" of a parent window) that intersects the parent window. Moving to state 308, the coordinate system of the visible region is translated from the parent window to the child window (so that the coordinates, e.g., Cartesian coordinates, of the click correspond to the relative origin specified by the child window handler) and the mouse down event is dispatched to the child window at a state 310. From

state 310, the kernel proceeds to state 312 to restore the coordinate system back to the parent since the child window processing is complete and the function is terminated at end state 292.

Now referring to Figure 11, the function to handle clicking in a parent window not having an active child document (see state 298 in Figure 10), called TDocumentWindow::DoMouseDown, is entered by the kernel at a start state 316. Transitioning to a decision state 318, a test is made on whether the click was on an inactive (or "normal") linked subscription, or subscriber document. (A more detailed discussion of the state 318 will be made hereinbelow with reference to Figure 13.) If not, the function terminates with no action at an end state 320. Otherwise, the kernel moves to a decision state 322 to test whether the child window is being dragged. This decision is affirmed if the mouse was moved beyond a preselected distance or if a second click was not received within a preselected time period. With a positive response from decision state 322, the kernel moves to a state 324 and proceeds to move or "drag" the child window. A "drag" results in a border being drawn around the child and the child window and border being moved with the pointer 200 (Figure 5). The kernel then terminates the function at state 320.

If it was determined in decision state 322 that no drag occurred, the kernel moves to a decision state 326 to test for a double click. If no double click, i.e., a quick succession of button depressions on the mouse 118 (Figure 1), was detected, the kernel proceeds to state 320 to terminate the function.

From decision state 326, if a double click on the inactive linked subscription was received then the child document is "started" at state 330. In starting the document, the child is made "live" or active and the creator module is initiated, e.g., the paint module if the child document contains graphic objects that were created by the paint module. (A more detailed discussion of the state 330 is made

hereinbelow with reference to Figure 14.) The kernel then terminates the function at end state 320.

Figure 12 is a screen display that is displayed on the video display 124 (Figure 1) after the child document, or subscriber document 192, is made active by double clicking on the inactive linked subscription of Figure 5 (no window is open for such a document), and after a portion of the paint object 193 has been deleted by the user. The tool bar 196 has changed to reflect the operations that are made available by the paint program. Also, the pointer 200 has changed shape from an "I-beam" pointer to a "pencil" pointer to indicate that paint objects may now be edited.

A child window 334 includes the window frame that is characteristic of a window displaying a child document. The window frame of the child window 334 is visually different from the window frame of the parent, or subscribing document, window 190 so as to provide a visual cue to the user that the editing of the child document 192 will be "in the context of" the parent document 194. For example, the child window 334 is bounded by a dotted line (instead of a solid line), the title bar is stippled (instead of containing "racing stripes"), and there is no shadowing as shown around the border of the active window 142 in Figure 2. Thus, the user is notified that the documents in the windows 190, 334 in Figure 12 are not completely independent as are the documents in the windows 140, 142 in Figure 2.

One skilled in the relevant technology will readily comprehend that documents may be embedded within in a document that is embedded in another document and so on. Therefore, the ICE of the present invention is designed to handle these recursive structures and the parent window does not necessarily have to correspond to a standard desktop interface window.

Returning to the description of the portion of the kernel that activates a child window (such as the window 334 in Figure 12), Figure 13 diagrammatically illustrates the control flow for "TSubscriptionList::HitTestFloaters"

(TSubscriptionList is a subclass of TObject). The test for floaters function of Figure 13 corresponds to the decision state 318 in Figure 11, i.e., before activating the child window, the ICE must decide which child document was clicked, if any.

The function of Figure 13 is entered at a start state 340 and the kernel proceeds to a decision state 342 to test whether there are more embedded (or child) documents in the parent document. Thus, there may be multiple child documents embedded in a parent document and the documents are sequentially searched for a hit until one is found. If there are no more child documents to process, the kernel terminates the function at an end state 344.

Otherwise, the kernel continues to a decision state 346 wherein it tests for whether there are more visible pages of the parent window. A document comprises one or more pages and the size of a page depends on the creator module. For example, the user may have chosen a printer record for the module that defines an 8-1/2" x 11" page size. The visible pages of the document are those pages that are included by the window (which can be resized at the user's option just as in the standard desktop interface). A child window is "nailed" or assigned to a particular page of the parent document. Thus, only the visible pages need be searched for a child document that may have been clicked and processing time is thereby conserved. The following describes the intersection processing associated with visible pages and child document pages.

At state 346, if all possible visible pages of the parent window have been checked, then the kernel proceeds back to state 342 to get the next embedded document, if one exists. Assuming that there is another visible page to be checked, the kernel moves from the decision state 346 to a decision state 348 to test whether the current child document is nailed to a current visible page of the parent window. If it is not, then the kernel returns to state 346 to select the next visible page.

On the other hand, if the test for a current child document being nailed to the current visible page is successful, at state 348, the kernel moves to a decision state 350 to determine whether the child document is in the "live", or active, state. If the child document is live, then the kernel moves to state 352.

The next set of processing states relate to the fact that the ICE uses a separate window strategy for child windows than that of the standard desktop interface. To accomplish the processing of the ICE, the kernel must trap all clicks to the parent window before making them available to the child windows. The trapping mechanism is realized by always making the child window not visible to the operating system, and selectively setting the child window to visible once the kernel decides that the child window should receive the click.

Hence, at the state 352, the visible region (only the portion of the document that is viewed through the window) of the child document is opened. Moving to state 354, the kernel gets the child's window handler and next, at state 356, the mouse down event is dispatched to the child window via the selected handler to determine what portion of the child window received the click. Once the mouse down event has been handled by the child window handler the visible region of the child document is closed (state 358).

Returning in the discussion to the decision state 350, if the child window is not active, the kernel proceeds to state 360 to calculate the corner coordinates of the contents portion of the inactive window. Now, the kernel moves from either of states 358 or 360 to a decision state 362 to decide whether the click occurred in the child document (whether active or inactive). If the click did not occur in the contents area, (i.e., the click was in a visible page of the parent document that contained a child document, but the current child document was not under the pointer 133), the kernel proceeds back to state 346 to check for more visible pages.

If, however, it is determined at state 362 that the click

is in the contents area of the child window, or in the child document if the child is not "live", the kernel proceeds to state 366 to create a unique token for the child document. The token is used internally by the kernel to identify the hit child document. At present, the token is not made accessible to a module as, in keeping with the gestalt of the invention, a module need not have knowledge of other documents. After the token is created, the kernel terminates the function at an end state 368.

Figure 14 illustrates the control flow for the "TSubscriptionList::LaunchPublisherFromToken" function which is a part of the kernel. More specifically, the function of Figure 14, which is entered at a start state 372, corresponds to the start document state 330 of Figure 11.

Proceeding to a decision state 374, the kernel therein decides whether an intermediate file (for instance, the intermediate file 178 shown in Figure 4), containing the subscriber, or child, document (e.g., 179, Figure 4), exists. If the intermediate file for the presently selected child document cannot be found in the computer 100 (typically on the hard disk 114), the kernel moves to state 376 and causes a dialog box, which is a feature of the standard desktop interface, to be presented on the video display 124 to prompt the user for further information of the whereabouts of the intermediate file at state 376. Moving to a decision state 378, if the intermediate file still cannot be located, the kernel terminates the function at an end state 380, and the user is not allowed to edit the child document. This situation may occur if the user has previously broken the subscription link by, for example, deleting the intermediate file.

More typically, at state 374, the intermediate file will be found and thereafter opened at a state 382. Likewise, if the intermediate file is found while in state 378, the kernel will move to state 382 and open the intermediate file. For example, in Figure 5, the file containing the map object 193 is opened. Continuing to a state 384, the module and file

references (e.g., 182, 184, Figure 4) are read by the kernel. The module that created the child document, for instance, the paint program used to create the map object 193 (Figure 12) is opened at state 386. The window (e.g., 334 in Figure 12) is then opened at state 388 and the child document is opened and the opened document is made available for viewing through the window which is opened at state 390 by the kernel. Moving to state 392, the publishing document is scrolled by the kernel to the subscriber document portion that is embedded in the parent, or subscribing, document. Then, proceeding to state 394 the child document is marked as "live" and the intermediate file is closed at state 396. The intermediate file is closed here so that other windows may access the same file since the intermediate file may be shared by more than one subscribing document. Lastly, the kernel terminates the function at the end state 380.

Figure 15 illustrates the control flow for the "TSubscriptionList::DrawFloaters" function that is a part of the kernel. The function illustrated in Figure 15 is executed by the computer 100 (Figure 1) when the user clicks the mouse 118 and the pointer (e.g., 200, Figure 12) is located at a box or arrow in a scroll bar.

Entering the function of Figure 15 at a state 400, the kernel continues to a decision state 402 to decide whether embedded, or child, documents exist in the window that received the click. If no child documents exist, or the child documents have all been processed, the kernel terminates the function at an end state 404. Otherwise, the next child document in the list of child documents owned by the window is selected and the kernel moves to a decision state 406.

A discussion of processing, i.e., a sequence of states, similar to the following processing, that relates to determining the intersection of visible pages and child pages, was presented above with respect to Figure 13.

At the decision state 406, the kernel tests whether there are more visible pages in the parent window that must be checked to determine whether the hit occurred in a child

window. If there are not any more visible pages to check, the kernel returns to state 402 to get the next child document. Alternatively, the kernel proceeds to a decision state 408 to test whether the page associated with the child document is the same as the currently selected visible page of the parent window. If the test is unsuccessful, the kernel returns to the state 406 to select the next visible page.

Assuming, on the other hand, that the child document is in a visible page, the kernel moves from state 408 to a decision state 410 to determine whether the child document is in the "live" state. If the child document is live, the kernel proceeds to state 412 to move the child window according to the amount of scrolling in the parent window that was triggered by the click in the scroll bar. Next, at a state 414, the kernel erases the region of the parent window behind the child window. Proceeding to a state 416, the kernel calculates the intersection of the parent visible region and the child window, since, for example, if the child window overlaps the parent window, only a portion of the child window will be drawn. After the intersection calculation, the kernel moves to a state 418 wherein the window frame elements of the child frame (or window) are drawn on the video display 124 (Figure 1). The kernel next moves to a state 420 wherein, as a final step in drawing the active child window after a scroll, the content area of the child window is drawn on the video display 124.

Referring back to decision state 410, if the child document is determined to be in the "normal" state (i.e., the linked subscriber document is inactive), the kernel moves to a state 422 wherein the child document is drawn inside of the parent window on the video display 124 (state 422). The kernel then continues to a decision state 424 and queries whether a border should be displayed around the selected child document. A border, such as a dashed line, may be drawn around the child document if, for example, the user has chosen a "Show Borders" operation in the "Edit" menu. If the border is not required the kernel moves back to the state 406 to

check the next visible page for child documents. If it is determined in state 424 that a border is required, the kernel moves to a state 426 and draws a border around the child document. The kernel then returns to the state 406 to check for more visible pages.

Turning to Figure 16, a screen display illustrates the scrolling aspect of the present invention as performed by the function shown in Figure 15. The user has dragged the scroll box 156 down the vertical scroll bar 150 and the child window, or floater 334 has moved in the context of, or along with, the parent window 190.

Figure 17 illustrates the control flow for the "TDocument::CloseLink" function of the kernel. The function of Figure 17 is executed by the computer 100 (Figure 1) after the child window is caused to be closed, for example, the user double clicks in the content area of the parent window that is outside of the active child window.

In Figure 17, the kernel enters the function at a start state 430 and proceeds to a state 432 wherein the active child window is marked as invalid. By marking the window invalid, the kernel initiates an update sequence of the parent window by dispatching an event to the window handler which causes the window to be redrawn. Moving to a state 434, the kernel closes the active child window (e.g., the window 334 in Figure 12). The kernel then moves to a state 436 and saves the contents of the child window, which have possibly been modified as has, for instance, the map object 193 of Figure 12. The contents of the child window are save backed to the intermediate file 178 (Figure 4) on the hard disk 114 (Figure 1). As a final step, the kernel moves to a state 438 and signals the parent window to read the new intermediate file so that on the next display cycle the modified child document will be displayed without a window frame. The kernel then moves to state 440 and terminates operation of the function.

Figure 18 is a screen display illustrating the effect of closing the child window 334 shown in Figure 12. The parent window 190 now displays a composite document comprising the

parent document containing text objects and the child document containing the graphic object 193 which was edited to remove undesired portions.

Figures 19 to 29 illustrate the function and results of the frame tool portion of the presently preferred in-context editor (ICE) of the present invention. The frame tool provides a means for creating a linked, subscriber document of a selected object type while a subscribing (or embedding) document is simultaneously displayed. In particular, the discussion hereinabove made reference to a second document already in existence prior to it being linked with a first, displayed document. However, the following discussion of the frame tool shows that the present invention also permits a second or subscriber document to be embedded in the subscribing document at the same time that it is created.

Initially referring to Figure 19, the parent window 190, displayed on the video display 124 (Figure 1), is shown to contain the text objects of a sample, first document which may be modified by a word processor program or module. The tool bar 196 includes word processor specific tool buttons, such as the superscript button indicated at 458, so that the active document in the parent window 190 can be operated on by the user (not shown). To create an embedded, second document, a frame tool button 460 in the tool bar 196 has been clicked using the mouse 118 of Figure 1 thus causing the ICE to enter a frame mode. In the frame mode, the ICE opens a rectangular border or frame 462 defining a region of the window 190 that has been positioned and sized by the user with the mouse 118.

Figure 20 illustrates another screen display, presented on the video display 124 (Figure 1), that continues the frame tool example initially described with respect to Figure 19. In Figure 20, the ICE requests that the user select an object type, or module, using a dialog box 464. The dialog box 464 contains a set of buttons 466 having icons that are indicative of the modules, such as word processor, that may be selected by the user. For instance, the button 466a may be clicked to select the manipulation of number objects by a spreadsheet

module. Although the box 464 only shows spreadsheet 466a, database 466b, draw 466c and paint 466d buttons, it will be understood that other object types or modules may be made available to the user. Once one of the user selectable buttons 466 has been clicked, the user clicks on a "New" button 468 to create a new document that will be associated with the requested module. The new, blank document will then be at least partially displayed inside the region inside of the frame 462.

Figure 21 is a screen display which continues the example of the frame tool from Figure 20 and shows that two documents are now simultaneously displayed albeit one document is blank. The parent window 190 now contains the child window 334 and the tool bar 196 comprises a new set of tool buttons which includes tools offered by the draw module, since the active window 334 presents a second document that specifies draw objects. As an example, a button 472 can be clicked if a new rectangular object is desired to be drawn inside of the second document. Note that at the time of the display shown in Figure 21, three independent files exist on the hard disk drive 114 of the computer 100, one file containing a text document and modifiable with the word processor module, an intermediate file, and another file for a draw document that will contain draw objects which can be created and modified with the draw module.

The function of the frame tool, as so far discussed, will be described with reference to the flow diagrams of Figures 22 to 25. Figure 22 illustrates the mouse down handling function TToolbar::DoMouseDown which handles a button, including the frame tool button 460, that was clicked in the tool bar 196 using the physical button on the mouse 118 (Figure 1).

Returning for the moment to Figure 9, a clicked button in the tool bar 196 (Figure 21) will be indicated to the kernel as a positive test at the decision state 278 indicating that the ICE has localized the click in the content portion of the display. The get window handler state 280, selects the TVisibleThing::HandleMouseEvent function and sends an event

accordingly. Thereafter, the function of Figure 22 is entered after it is determined that the click was inside of the tool bar 196.

After entry of the TToolbar::DoMouseDown function at state 480, in Figure 22, the kernel moves to state 482 and retrieves, from the memory 106 (Figure 1), the address of the live or active document structure. The kernel proceeds to state 484 to retrieve the list of all active tools and, at state 486, the next (or first in this instance) tool index in the active tool list is calculated. It should be noted that the tool list of all modules that belong to the ICE will normally include certain common tools, e.g., the frame tool (see Figure 19, button 460), and the module's own unique tools, e.g., the superscript tool (see Figure 19, button 458).

Proceeding to decision state 488, if it is determined that all tool buttons have been processed, that is, that the index of the current tool is beyond the last entry in the list of active tools, the kernel moves to the end state 494 and exits the tool bar handling function. However, of greater interest to the present discussion, if the kernel decides that all active tools have not yet been checked, the kernel continues to decision state 490 to test whether the currently indexed tool has a button that is visible and hit. If the indexed tool has a button that is not visible or not hit, the kernel loops to state 486 to get the next tool index (i.e., increment the index), otherwise, the selected tool button is handled at state 492 and the function terminates at the end state 494.

From the handle selected tool button state 492, after further processing of the tool button event by the TToolbar class, the TLocalKernel class receives the event and enters the handle create frame function, TLocalKernel::HandleCreateFrame, shown diagrammatically in Figure 23, at the start state 500. The kernel then moves to a decision state 502 to query whether the ICE is in "frame mode". If not, the kernel proceeds to state 504 to retrieve the address of the live document structure and subscription

list. Then, at decision state 506, a test is made for whether the window allows the use of the frame tool. For instance, it may be the case that a child window established to create a header of a footer in a text document is not allowed to embed another document using the frame tool. If true, the kernel sets frame mode "on" at state 508 and sets menu and tool bar states so as to highlight the frame tool button, terminating the function at state 512. On the other hand, if the test is negative at state 506, states 508 and 510 are bypassed and the function terminates at state 512.

Returning attention to decision state 502, if the ICE is already in frame mode and the frame tool button 460 (Figure 19) is clicked, then frame mode is turned "off" at state 514. The kernel moves from state 514 to state 516 to set menu and tool bar states, unhighlighting the frame tool button, and the function terminates as before at state 512. Thus, the frame tool button 460 serves to toggle the ICE in and out of frame mode.

Now turning to Figure 24, the mouse down handler function in TFrame is entered when the in-context editor (ICE) has frame mode set "on" (see previous Figures 22 and 23) and the user clicks the mouse 118 in the window 190. However, one will note that Figure 24 is an extension of the handler function already described with respect to Figure 11 so as to include the logic to handle the frame mode.

From the start state 316 in Figure 11, the kernel proceeds to decision state 520 in Figure 24 to query whether the ICE is in frame mode. The kernel continues as before through Figure 11, beginning at state 318, if the ICE is not in frame mode.

Otherwise, assuming frame mode is set "on", the kernel moves to state 522 to track a rectangle. More specifically, the position at which the mouse 118 (Figure 1) was initially clicked is stored as one corner vertex of a rectangular region within the outlined frame 462 (Figure 19). Now, the user moves the mouse 118 and the frame 462 contracts and expands since the first clicked corner is now "nailed" to one position

on the video display 124. The window 190 may even scroll if the user moves the mouse 118 so that the pointer moves into a portion of the document that is not presently displayed. At state 522, the user clicks at a location in the window 190 for the second, diagonal vertex of the desired frame 462 and the location is saved in the memory. Proceeding to decision state 524, the kernel checks whether the rectangular frame 462 is larger than a minimum size, and if it is not, then frame mode processing is exited to state 320 in Figure 11. For example, in one embodiment the minimum size rectangle is 25 pixels by 25 pixels. A minimum size is needed for the practical reason of manipulation by a mouse, e.g., inserting window controls such as a growth button, and for displaying an image that is meaningful, i.e., not so small that it cannot be read by a human.

Therefore, if a minimum size rectangle has been defined by the user, the kernel creates a document in frame event, at function 526, and sends the event to the document (state 528), e.g., the text document shown in Figure 19, and continues to state 320 in Figure 11. Basically, since the present or subscribing document and associated module, e.g., word processor, is known and active, the event is sent to itself, and the task of creating a new (e.g., second) document is handled at the appropriate time by the module. The presently preferred function 526 creates a document of a user specified object type, creates and saves a file, corresponding to the new document, to the hard disk 114, translates the coordinate system of the frame 462 to the upper left corner of the file (or publishing document), creates and saves an intermediate file to the hard disk 114, creates a link in the subscribing document with the intermediate file, and opens a child window as, for example, indicated at 334 in Figure 21.

The create document in frame function 526, entered by the kernel at start state 532, is more fully appreciated with reference to the control flow diagram of Figure 25. Moving to state 534, the kernel opens a dialog with the user to select a module. Figure 20 is a screen display showing an example of

the dialog, wherein the dialog box 464 is presented to the user with a list of module (or object) types 466. Then, at decision state 536, a test is made for whether a module was selected and if one was not selected, e.g., the user clicked on the cancel button 469 (Figure 20), the function is exited at end state 537.

Assuming that a module is selected, e.g., the draw module button 466c (Figure 20) is clicked, the kernel creates a new document which includes allocating a part of the memory 106 (Figure 1) for a document structure and opening a new file on the hard disk 114. Proceeding to decision state 540, if the creation was unsuccessful, for example, no memory could be allocated, for the document structure the function terminates through end state 537. Otherwise, the file definition structure owned by the document is retrieved from the memory 106 (state 542) and the document and file are saved to the disk 114 (state 544).

Continuing from state 544 to decision state 546, the kernel tests whether the file now exists on the disk 114. If so, the kernel moves to state 548 to translate the coordinate system of region in the frame 462 to the coordinate system of the newly created, publishing document. Then, at state 550, the kernel creates an intermediate file 178 (Figure 4) on the hard disk 114. The kernel next moves to state 552 to automatically create a link between the subscribing document, shown for example as text in the parent window 190 of Figure 21, and the subscriber document defined by the intermediate file. Lastly, at function 330, the function shown diagrammatically in Figure 14, TSubscriptionList::LaunchPublisherFromToken, is invoked. The start document function 330 opens a module (e.g., the draw module as shown in Figure 21), opens a child window (e.g., the window 334, Figure 21) and opens the subscriber document structure and file. The framed region is now the frontmost document on the video display 124 (Figure 1) and is ready to be edited.

As an alternative flow of control at the decision state

546, if a file is not found on the hard disk 114, the kernel moves to state 556 to delete the document structure from the disk 114. Then, at state 558, the document in the parent window 190 is made active and the function is exited at state 537.

As shown in Figure 26, assuming that the new document in the frame was properly created, the user can now create objects 562 in the subscriber document displayed in the child window 334 using well-known techniques of editing modules such as can be found in the draw module of the ICE. The procedure for handling user interaction with the document in the child window 334 has been previously discussed hereinabove but it is worthwhile to recap here in a summary form. Basically, a click event, caused by the user depressing the button on the mouse 118 (Figure 1) is fed by the kernel into the window handler for the parent window 190, i.e., the TDocumentWindow::DoMouseDown function shown diagrammatically in Figure 10. At decision state 288 it is determined that the parent window 190 is active and that the click was located in an active child window 334. The event is then dispatched to the child's window frame and handled by the TFrame::DoMouseDown function shown diagrammatically in Figure 11. At this point the user can edit objects or, for example, recursively enter frame mode again to create an embedded document inside the child window 334.

Referring to Figure 27, a click in the parent window 190 outside of the child window 334 (Figure 26) causes the child window 334 to not be displayed, and the documents are now viewed in the window 334 as a single, composite document containing text objects and draw objects 562 (although both documents retain their own unique identities in memory). Figure 28 shows that the child window 334 has been activated by a click, the draw module has been opened, and the user has modified the embedded document by deleting the circle object 563c (Figure 27).

The present invention includes an in-context editor with a frame tool. The frame tool allows a new document to be

created, positioned and object type selected while another document is being actively edited. Indeed, frames can be defined before objects are added, thus providing the equivalent of picture placeholders in a book. The frame tool frees the user from knowledge about the mechanics of linking documents.

While the above detailed description has shown, described and pointed out the fundamental novel features of the invention as applied to various embodiments, it will be understood that various omissions and substitutions and changes in the form and details of the illustrated invention may be made by those skilled in the art, without departing from the spirit and scope of the claimed invention.

WHAT IS CLAIMED IS:

1. In a computer, a method of editing a plurality of documents, comprising the steps of:

displaying at least a portion of a selected first one of the documents;

linking a selected second one of the documents to the first document;

displaying at least a portion of the second document simultaneously with said displaying of the first document;

positioning the second document at a selected location in the first document so that together the selected documents are displayed as a composite document; and

modifying the second document while the composite document is displayed.

2. The method defined in Claim 1, wherein the first document contains an object of a first type and the second document contains an object of a second type, the first type being different from the second type.

3. The method defined in Claim 1, wherein the second document is an intermediate document created from a selected portion of a selected third document.

4. The method defined in Claim 1, wherein the second document is linked to a selected third document such that when the third document is displayed the modified second document will be displayed simultaneously.

5. The method defined in Claim 1, wherein the modifying step includes displaying a window frame around the second document.

6. The method defined in Claim 1, additionally comprising the step of scrolling the first and second documents simultaneously.

7. The method defined in claim 1, wherein the first and second documents are modified by different modules which communicate with a kernel.

8. The method defined in Claim 7, wherein the kernel stores state and position information about the selected

documents and the modules store no information relating to the context of the documents.

9. The method defined in Claim 1, additionally comprising the step of storing the modified second document in the memory of the computer.

10. The method defined in Claim 1, wherein a viewable portion of the composite document is contained in a window frame.

11. A method of scrolling multiple windows in a computer, the method comprising the steps of:

displaying a first document in a first window;

displaying a second document in a second window wherein the second window is positioned inside the boundary of the first window; and

scrolling the first and second windows simultaneously.

12. The method defined in Claim 11, wherein the contents of the second window retain the same view of the second document both before and after the scrolling.

13. An in-context editing system having a kernel and one or more object specific editor modules, for computer-assisted modification of documents, the kernel comprising:

means for displaying document windows;

means for linking embedded and embedding documents;

means for determining a user selection of an embedded document displayed in a window containing the embedding document; and

means for passing user determined events to the module that created the embedded document.

14. The system defined in Claim 13, wherein the displaying means includes means for displaying overlapping windows.

15. The system defined in Claim 13, wherein the user selection is made with a mouse.

16. In a computer, a method of creating a composite document, comprising the steps of:

displaying at least a portion of a first document containing an object;

selecting a frame in the displayed first document portion;

selecting an object type from a plurality of object types;

creating a second document for containing an object of the selected object type which is displayable in the frame so that together the documents are displayed as a composite document;

linking the second document to the first document; and

creating objects of the selected object type in the frame while the composite document is displayed.

17. The method defined in Claim 16, wherein the first document contains an object of a first type and the second document contains an object of a second type, the first type being different from the second type.

18. The method defined in Claim 16, additionally comprising the step of scrolling the first and second documents simultaneously.

19. The method defined in Claim 16, wherein the first and second documents are modified by different modules which communicate with a kernel.

20. The method defined in Claim 19, wherein the kernel stores state and position information about the selected documents and the modules store no information relating to the context of the documents.

21. The method defined in Claim 16, additionally comprising the step of storing the second document in the memory of the computer.

22. The method defined in Claim 16, wherein a viewable portion of the composite document is contained in a window frame.

23. The method defined in Claim 16, additionally comprising the step of creating objects of the selected object type in the frame while the composite document is displayed.

24. The method defined in Claim 23, additionally comprising the step of modifying an object in the second document.

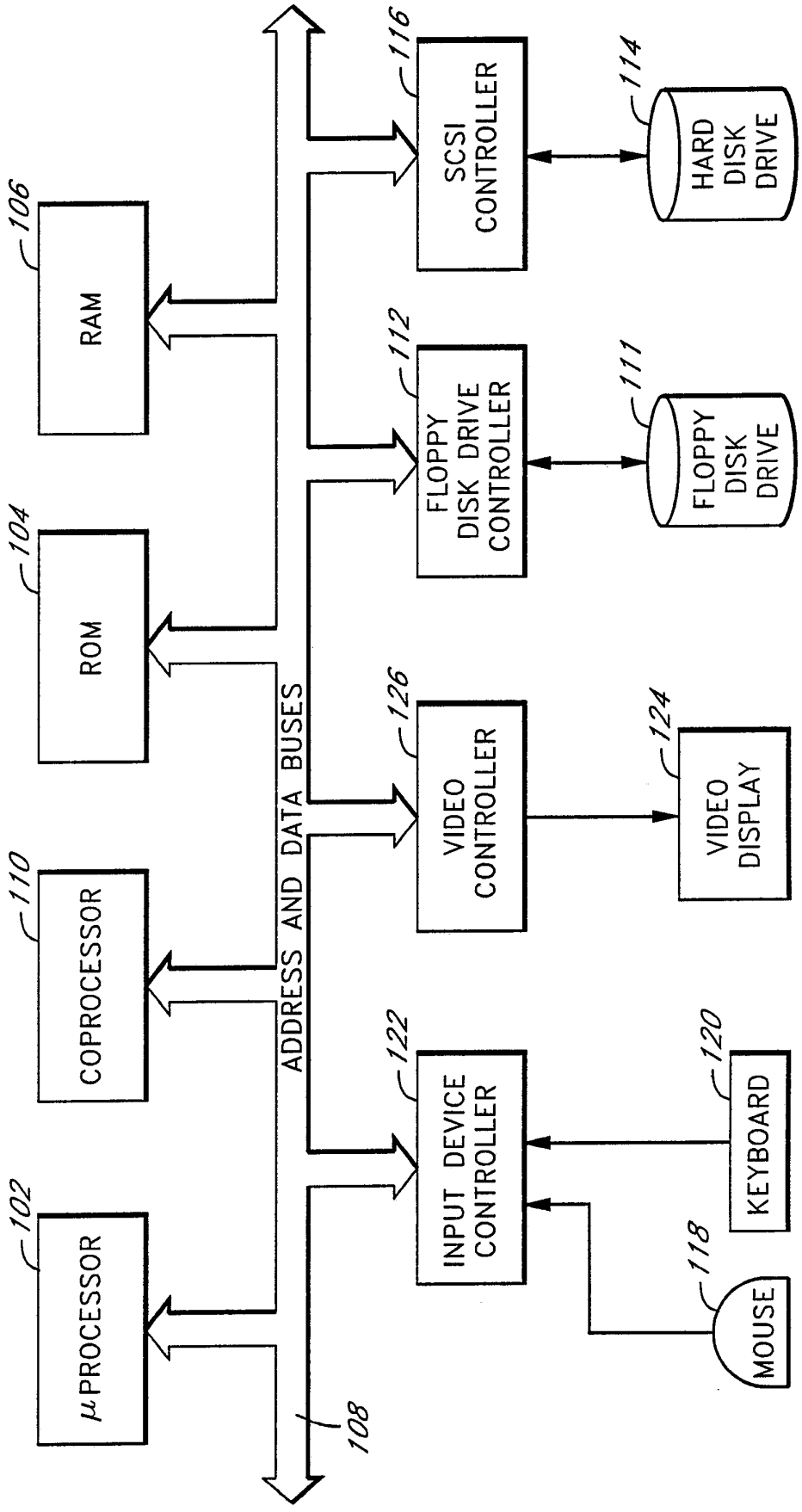

µPROCESSOR
102
COPROCESSOR
110
ROM
104
RAM
106
ADDRESS AND DATA BUSES
108
INPUT DEVICE CONTROLLER
122
KEYBOARD
120
MOUSE
118
VIDEO CONTROLLER
126
VIDEO DISPLAY
124
FLOPPY DISK DRIVE CONTROLLER
112
FLOPPY DISK DRIVE
111
SCSI CONTROLLER
116
HARD DISK DRIVE
114

FIG. 1

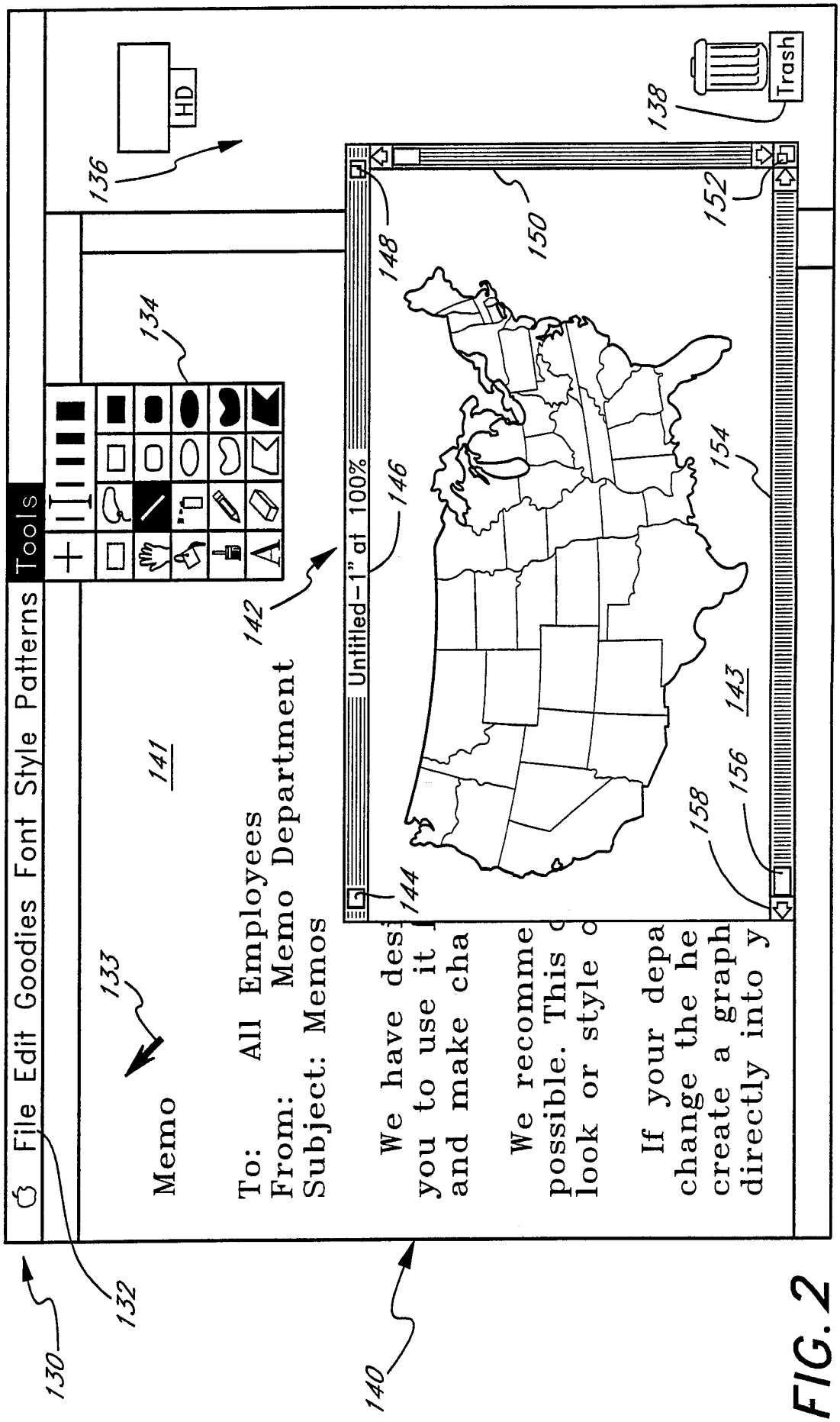

File Edit Goodies Font Style Patterns
Tools
HD
Trash
Untitled-1" at 100%
Memo
To: All Employees
From: Memo Department
Subject: Memos
We have desi
you to use it
and make cha
We recomme
possible. This
look or style
If your depa
change the he
create a graph
directly into y
130
132
133
134
136
138
140
141
142
143
144
146
148
150
152
154
156
158

FIG. 2

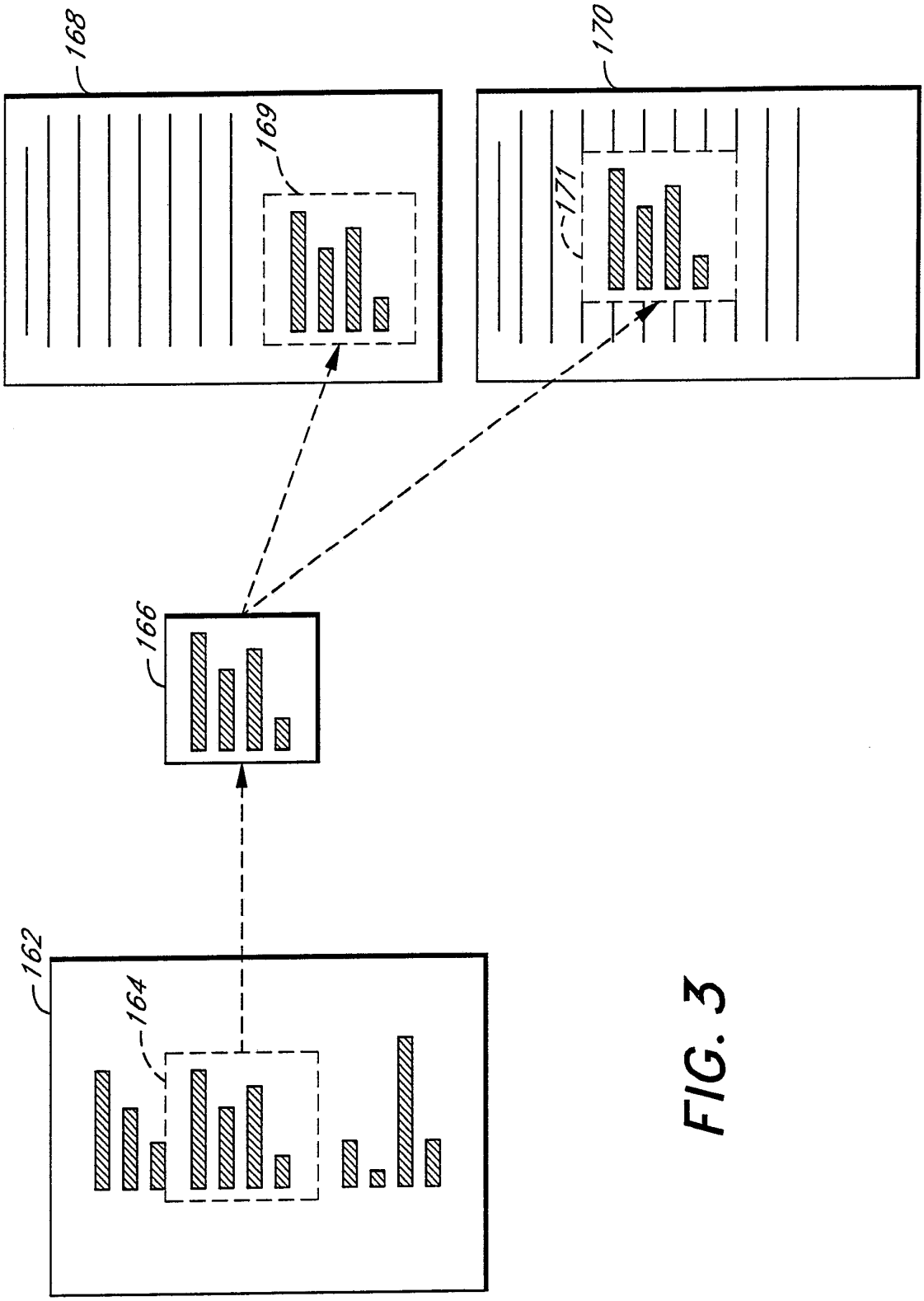



FIG. 3

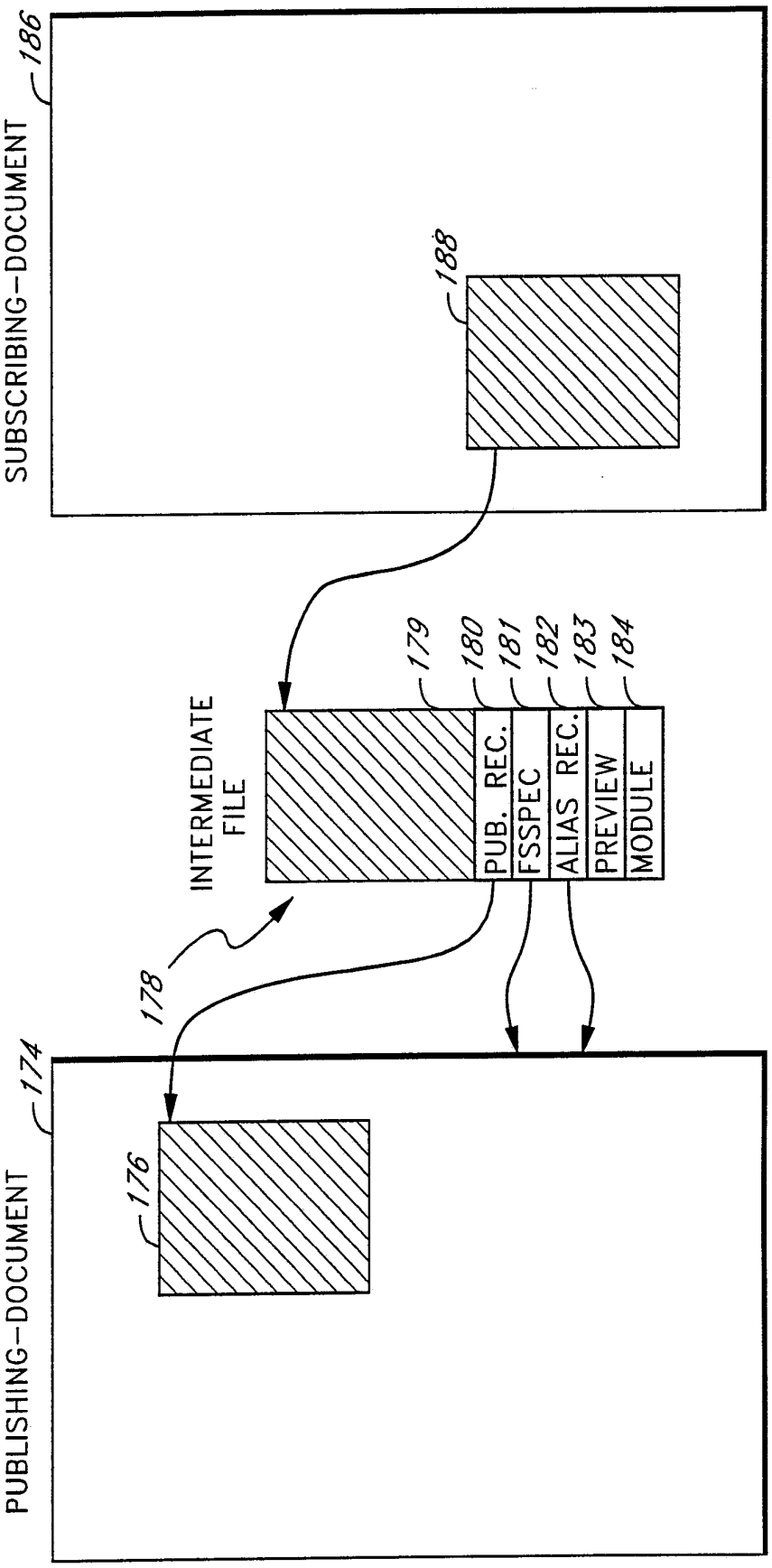

SUBSCRIBING-DOCUMENT
186
188
INTERMEDIATE FILE
179
180
181
182
183
184
PUB. REC.
FSSPEC
ALIAS REC.
PREVIEW
MODULE
178
PUBLISHING-DOCUMENT
174
176

FIG. 4

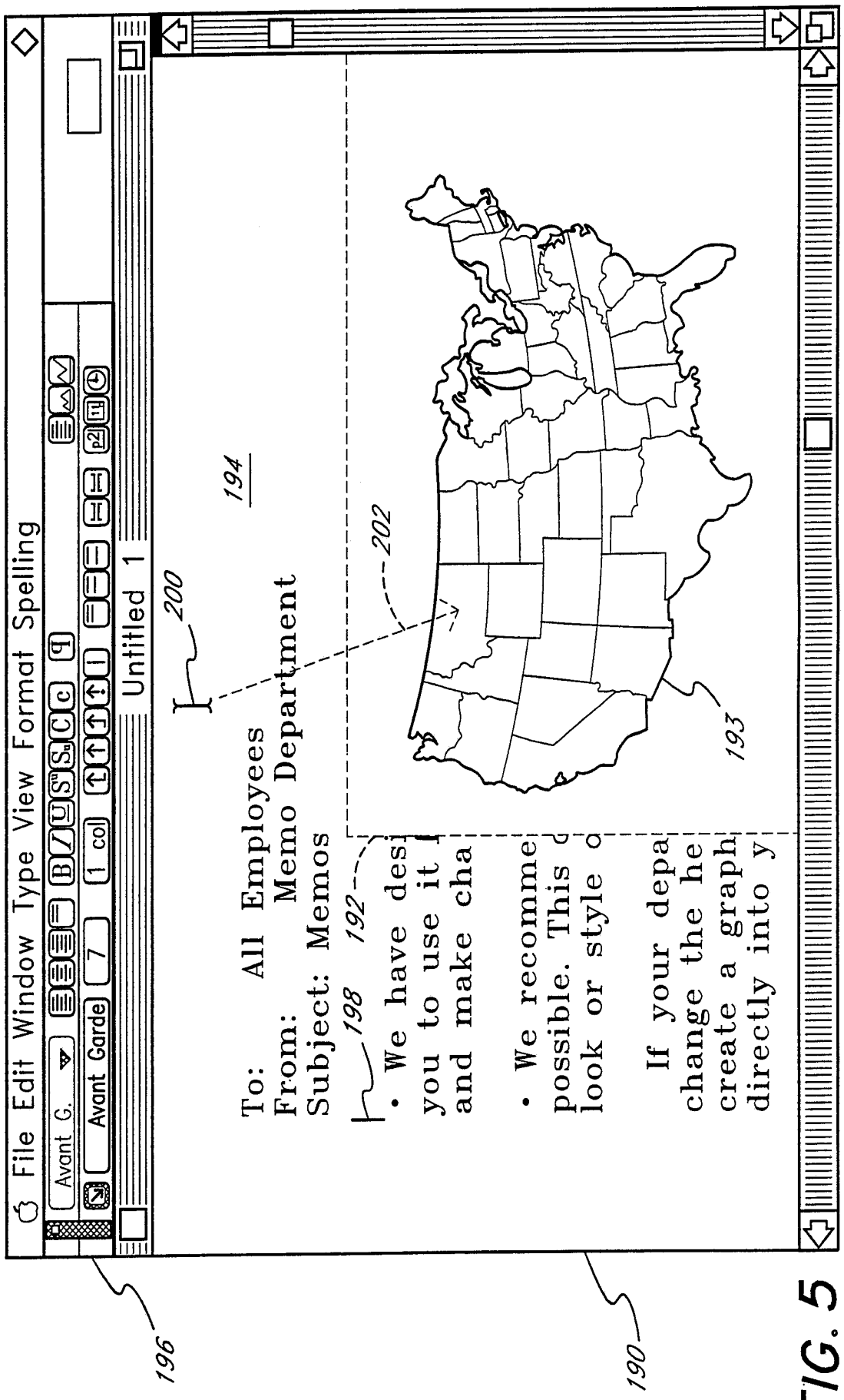

File Edit Window Type View Format Spelling
Avant G.
Avant Garde
7
1 col
Untitled 1
To: All Employees
From: Memo Department
Subject: Memos
• We have desi
you to use it
and make cha
• We recomme
possible. This
look or style
If your depa
change the he
create a graph
directly into y
194
200
202
198
192
193
196
190

FIG. 5

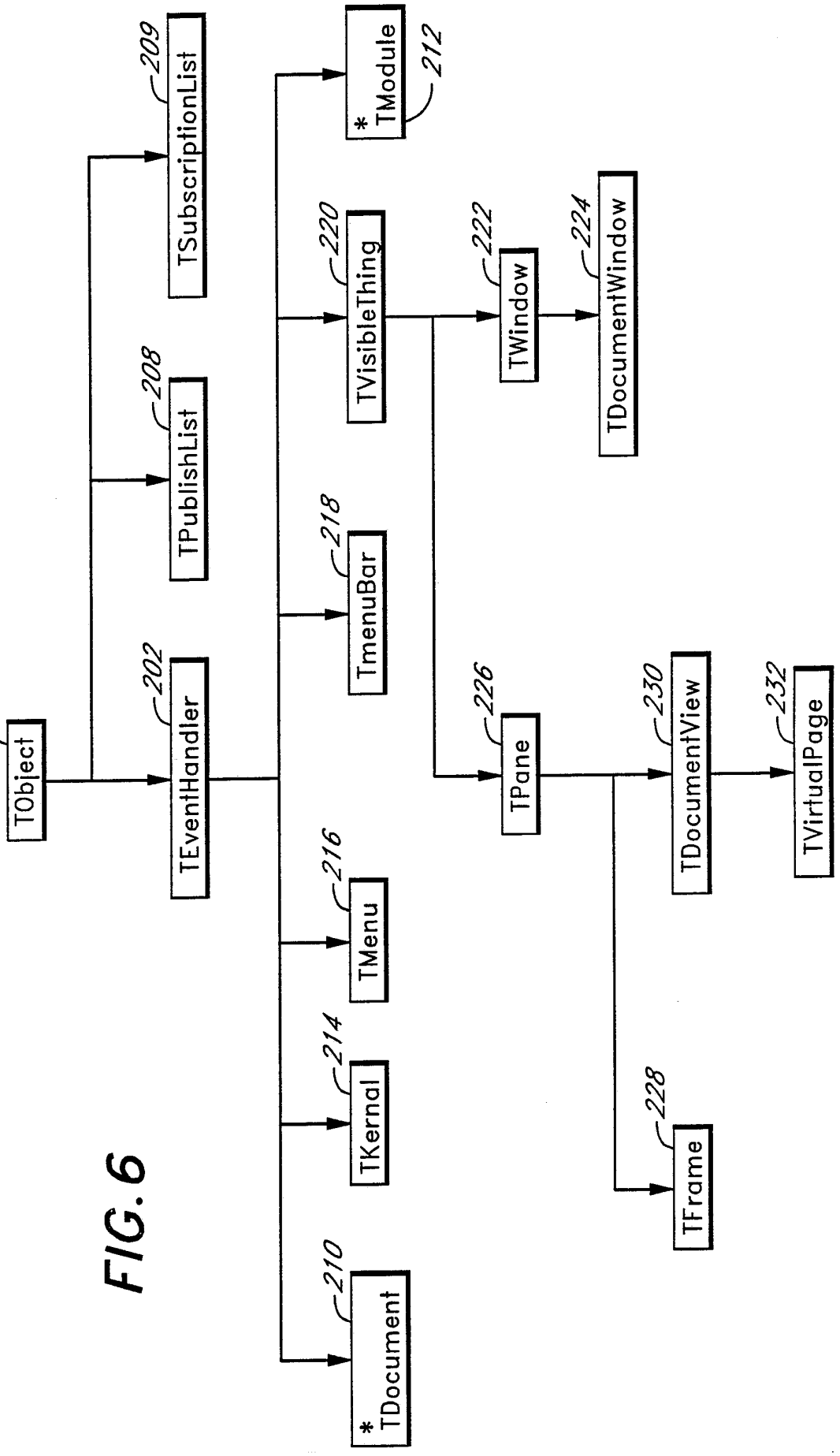

206
TObject
202
TEventHandler
209
TSubscriptionList
208
TPublishList
212
* TModule
220
TVisibleThing
222
TWindow
224
TDocumentWindow
218
TmenuBar
226
TPane
230
TDocumentView
232
TVirtualPage
216
TMenu
214
TKernal
228
TFrame
210
* TDocument

FIG. 6

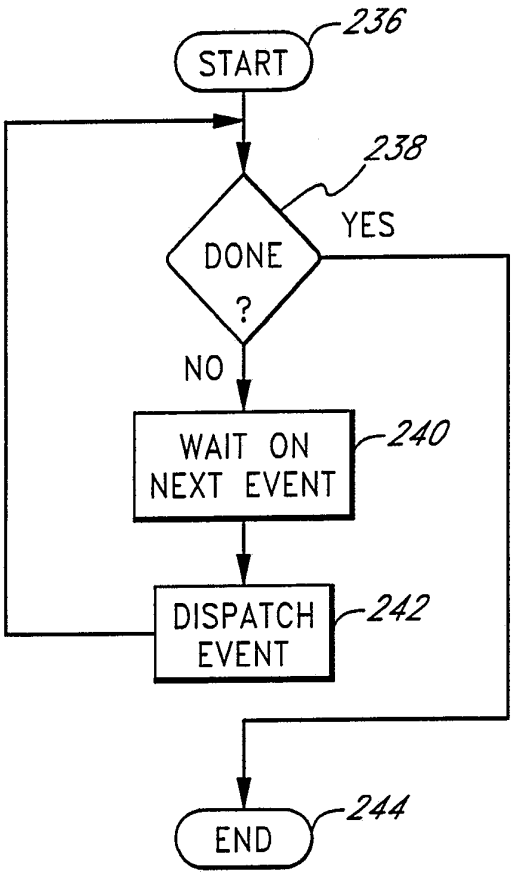



FIG. 7

TLocalKernal::EventLoop

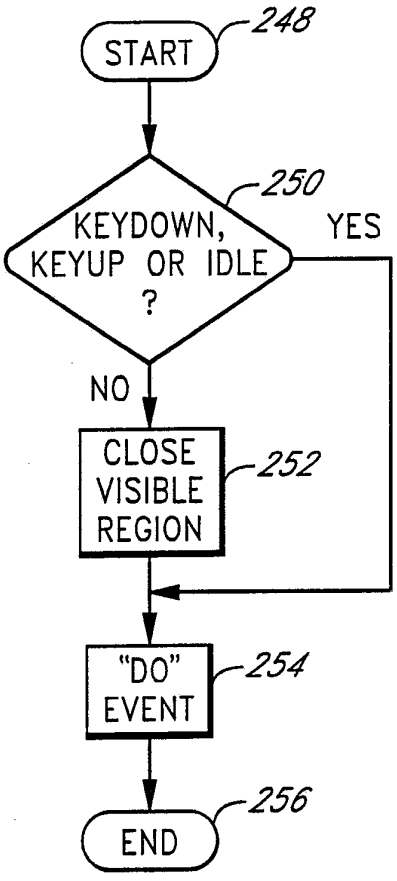



FIG. 8

TLocalKernal::DispatchOSEvent

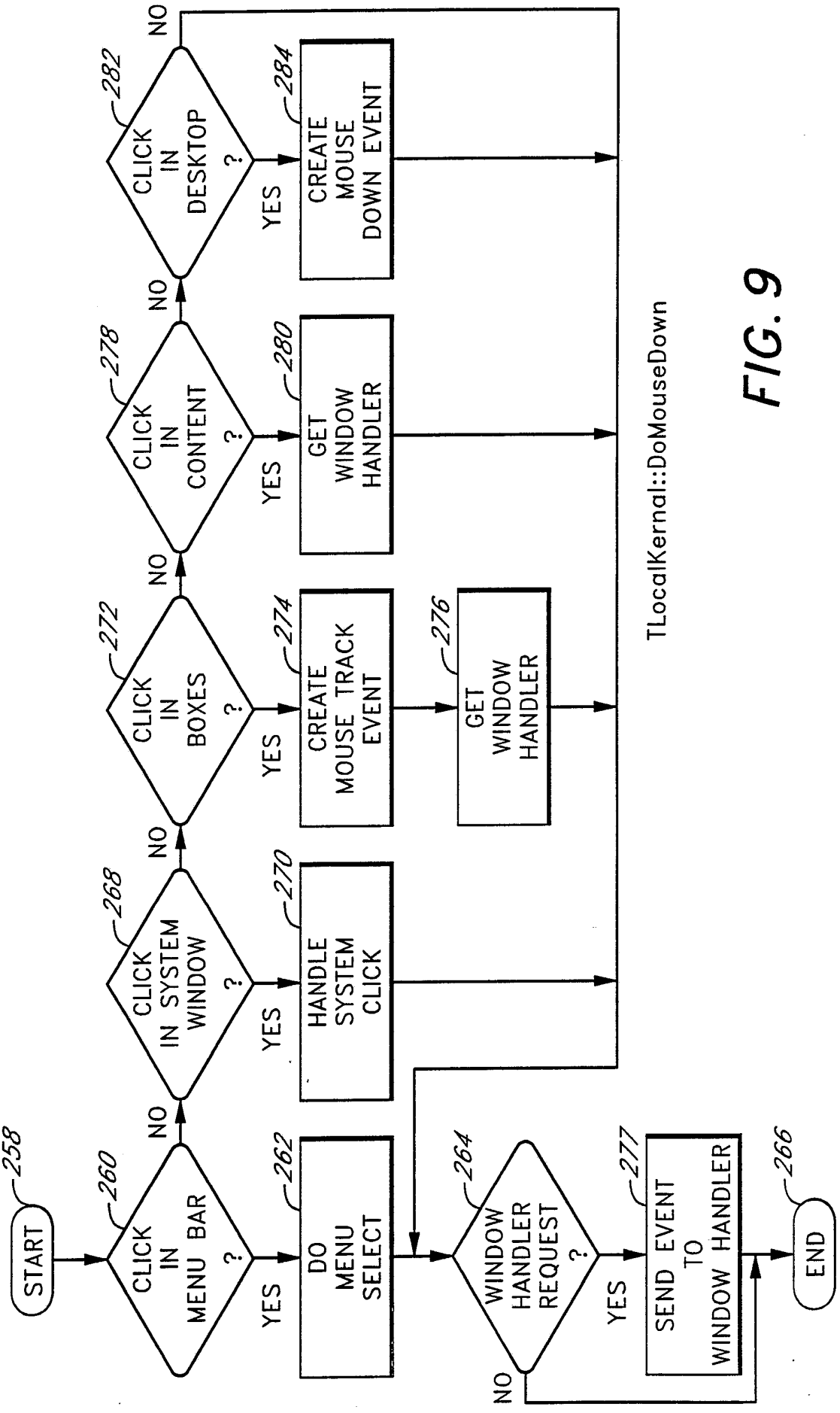
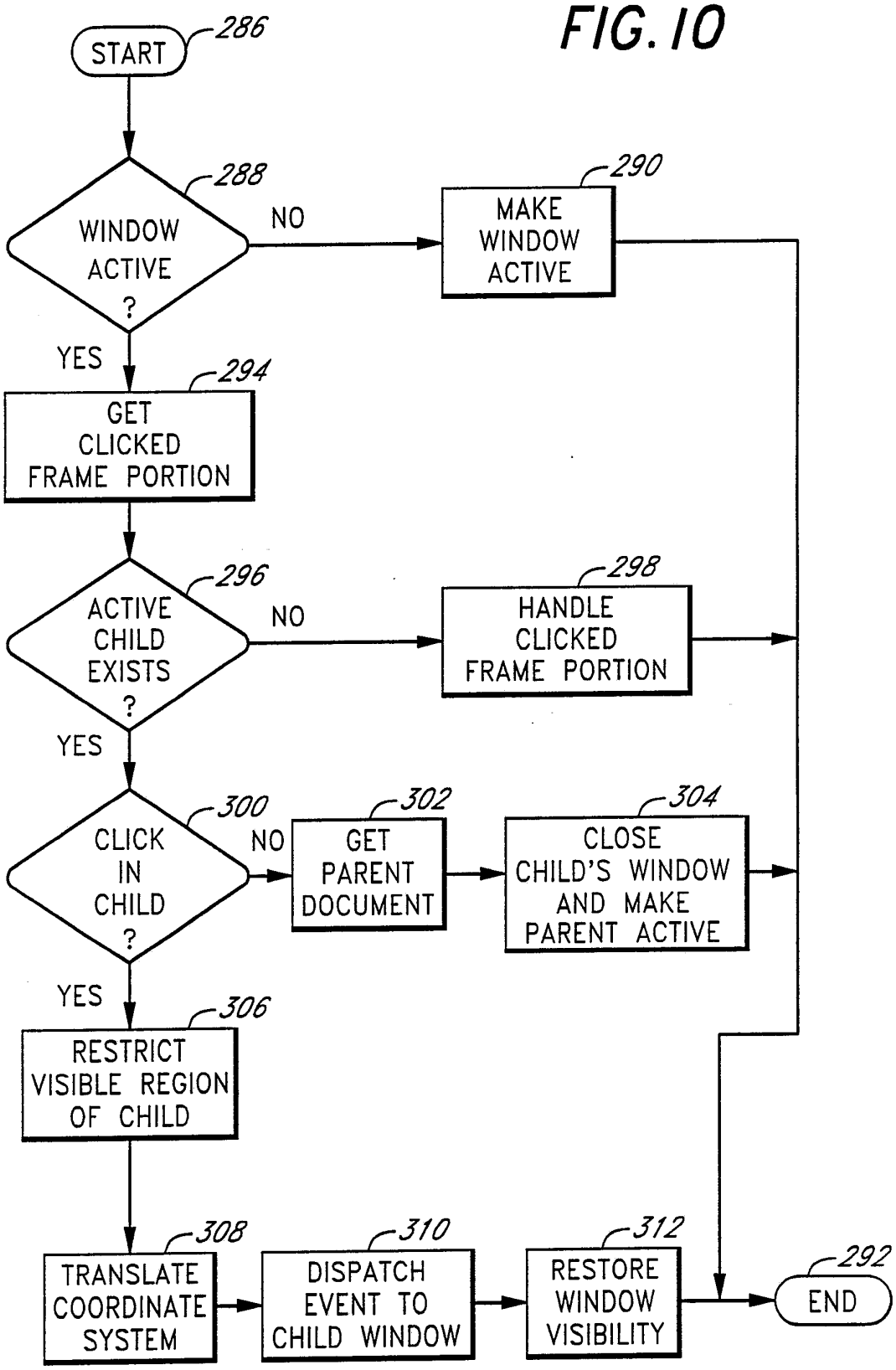

START
258
CLICK IN MENU BAR ?
260
YES
DO MENU SELECT
262
NO
CLICK IN SYSTEM WINDOW ?
268
YES
HANDLE SYSTEM CLICK
270
NO
CLICK IN BOXES ?
272
YES
CREATE MOUSE TRACK EVENT
274
GET WINDOW HANDLER
276
NO
CLICK IN CONTENT ?
278
YES
GET WINDOW HANDLER
280
NO
CLICK IN DESKTOP ?
282
YES
CREATE MOUSE DOWN EVENT
284
NO
WINDOW HANDLER REQUEST ?
264
YES
SEND EVENT TO WINDOW HANDLER
277
NO
END
266
TLocalKernal::DoMouseDown

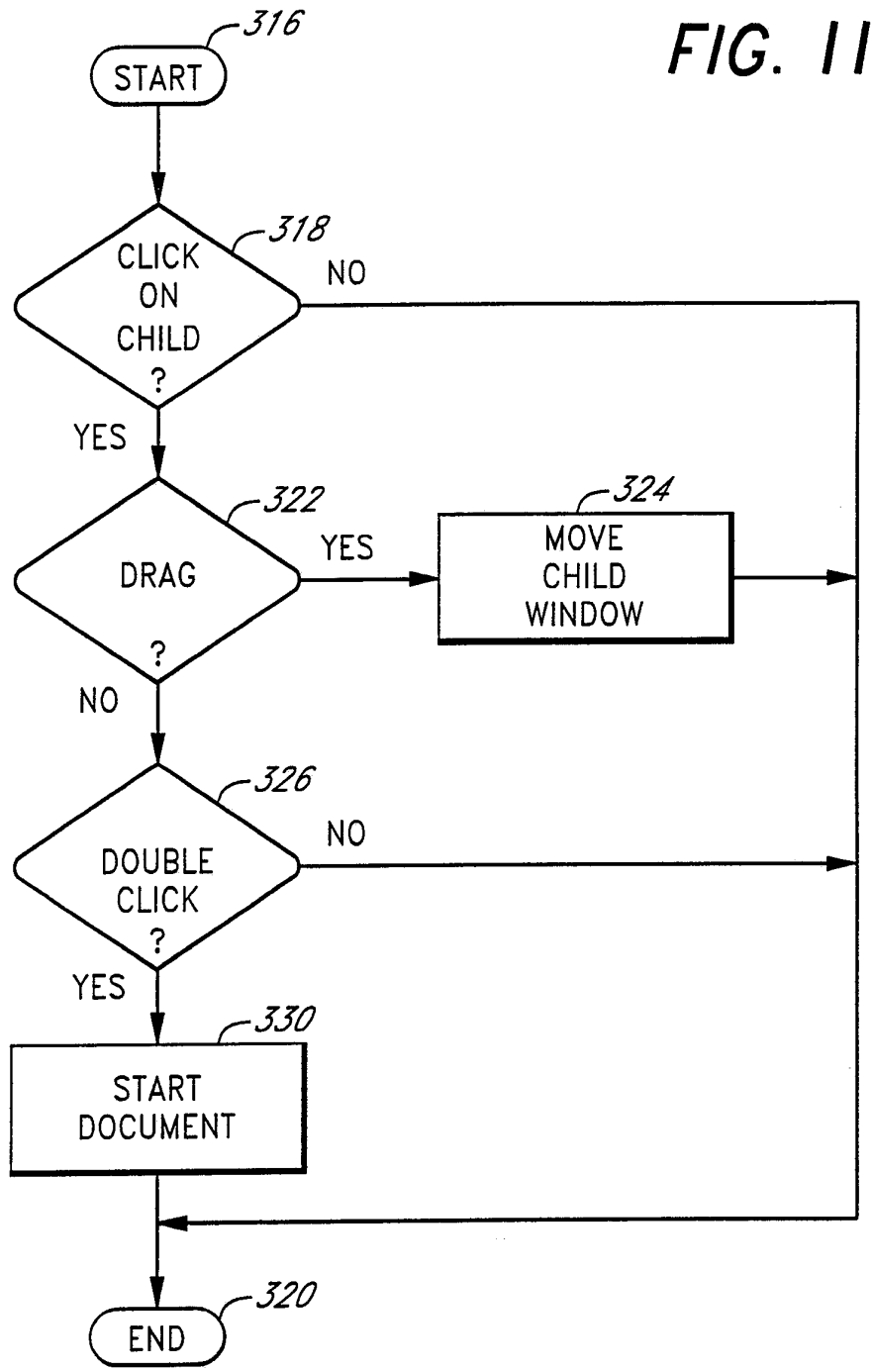
FIG. 9

FIG. 10




TDocumentWindow::DoMouseDown

FIG. 11




TFrame::DoMouseDown

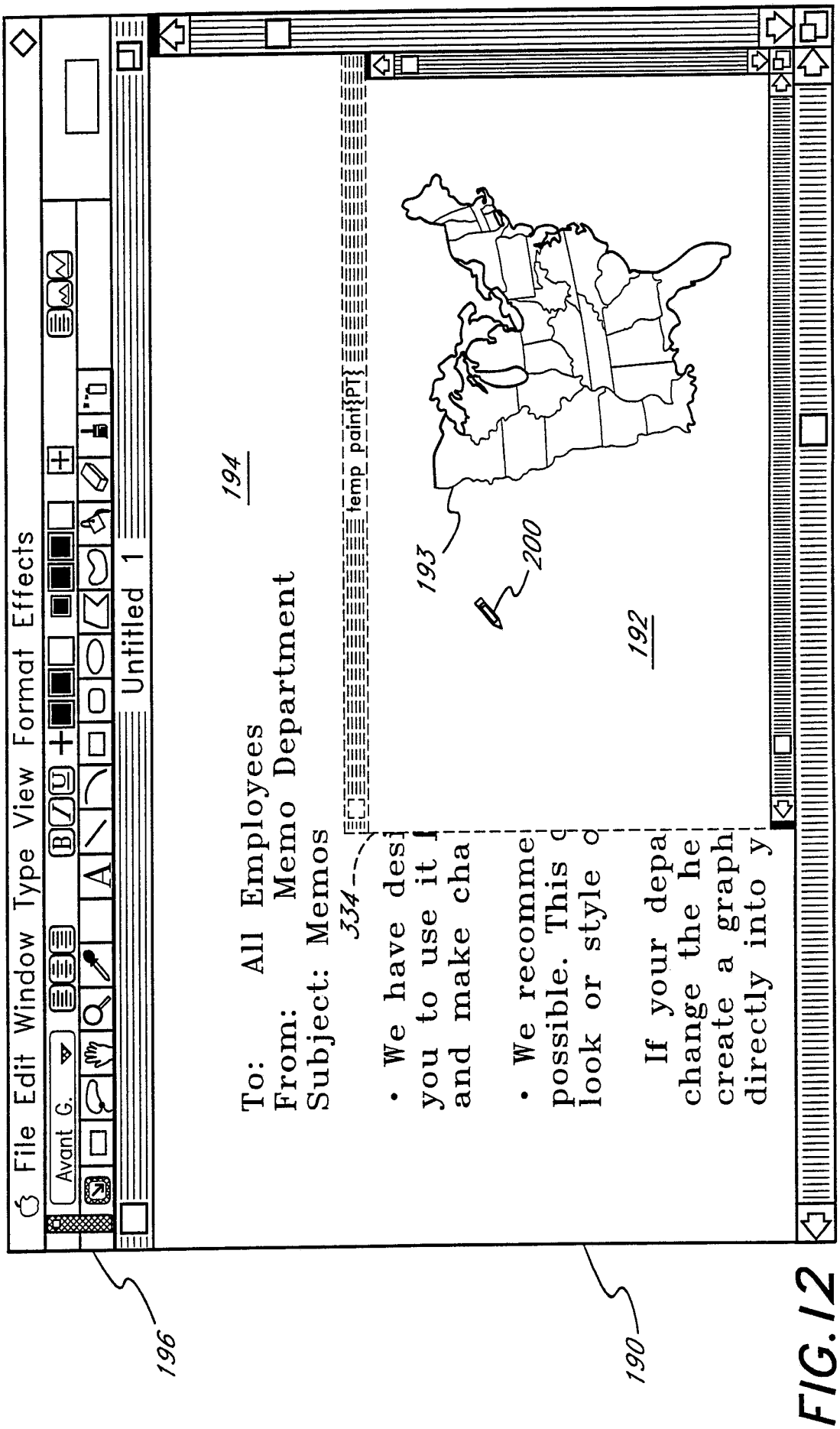
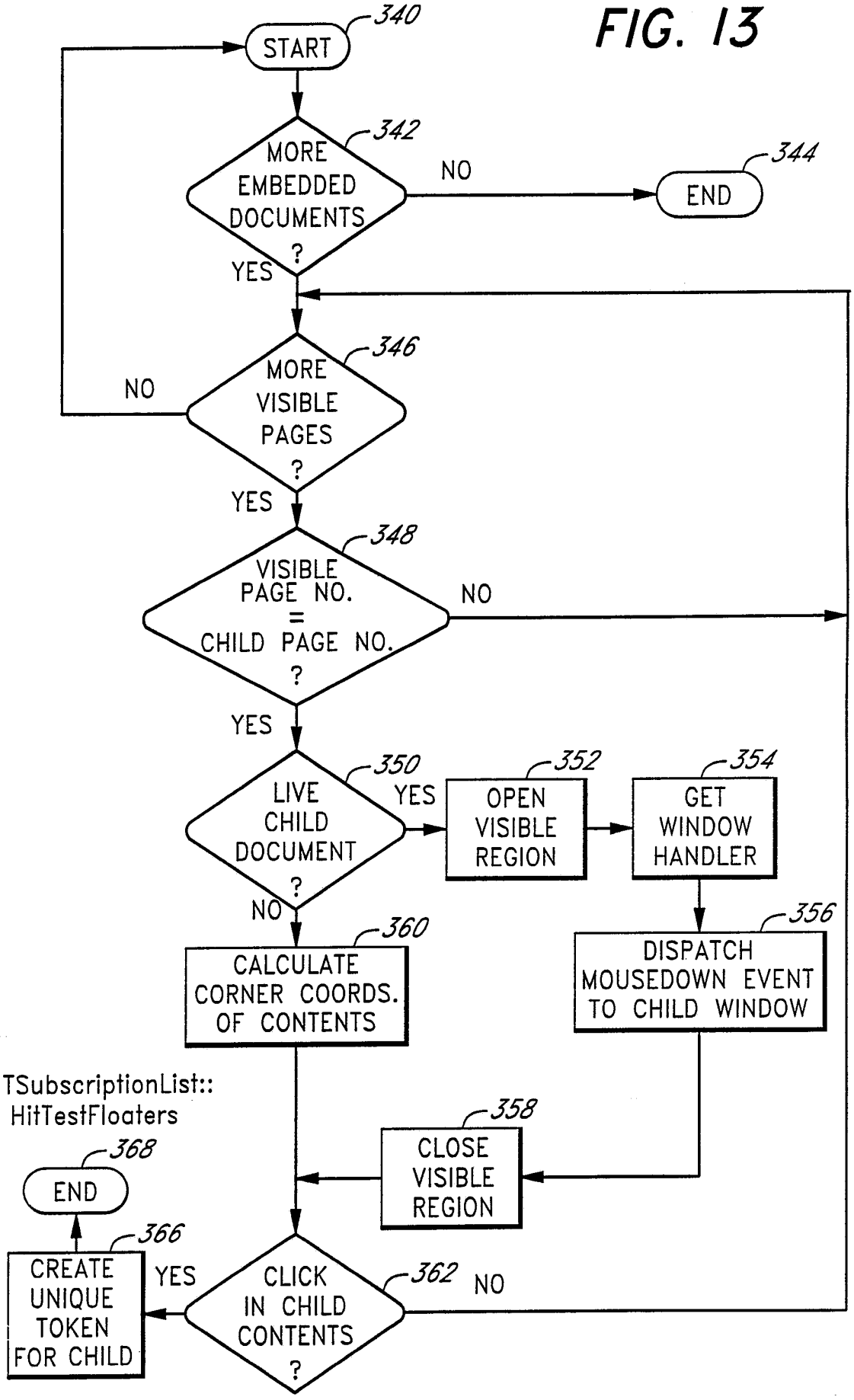
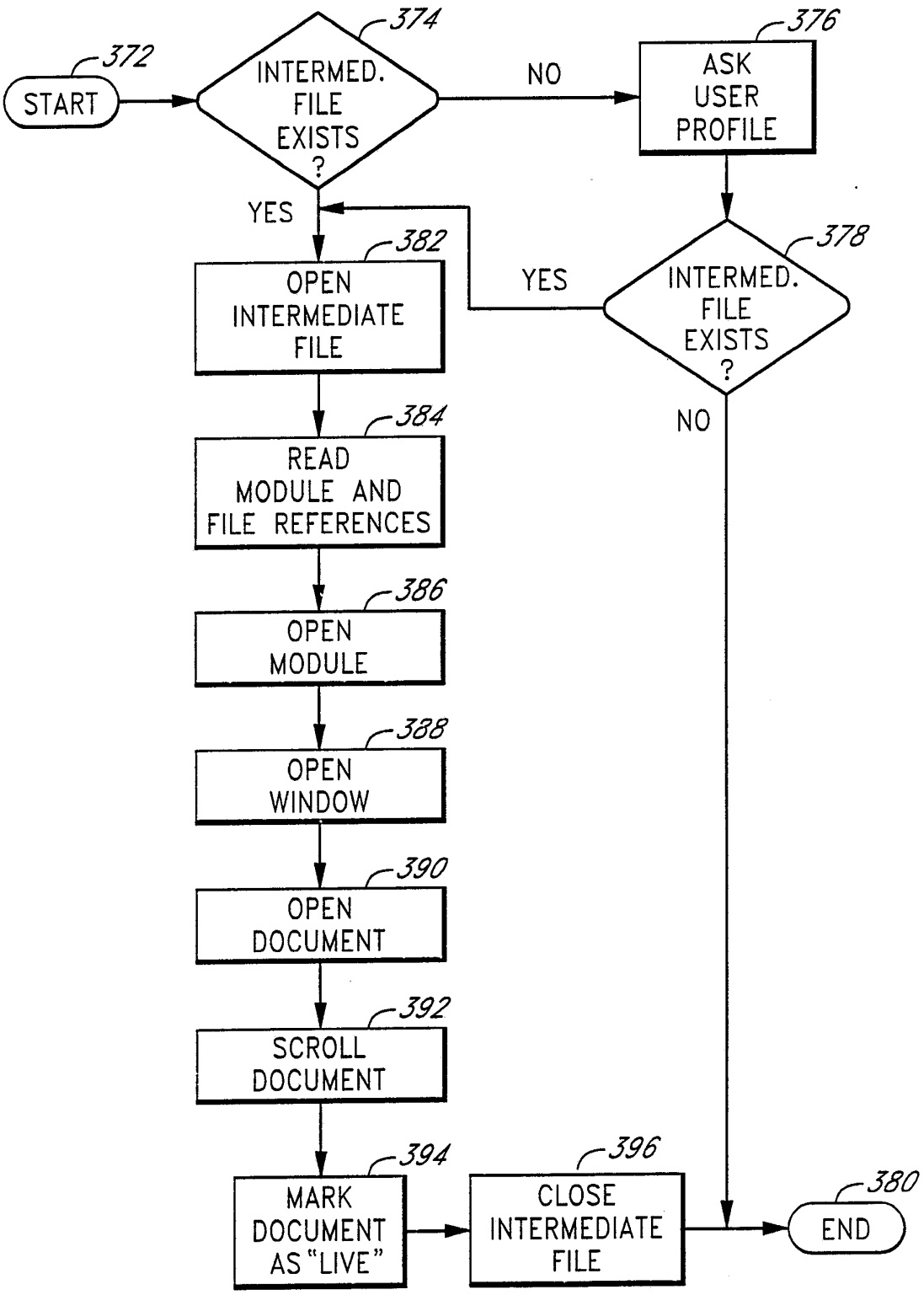

File Edit Window Type View Format Effects
Avant G.
Untitled 1
temp paint[PT]
To: All Employees
From: Memo Department
Subject: Memos
• We have desi
you to use it
and make cha
• We recomme
possible. This
look or style
If your depa
change the he
create a graph
directly into y
194
334
193
200
192
196
190

FIG. 12

FIG. 13


340
START
342
MORE EMBEDDED DOCUMENTS ?
NO
344
END
YES
346
MORE VISIBLE PAGES ?
NO
YES
348
VISIBLE PAGE NO. = CHILD PAGE NO. ?
NO
YES
350
LIVE CHILD DOCUMENT ?
YES
352
OPEN VISIBLE REGION
354
GET WINDOW HANDLER
NO
360
CALCULATE CORNER COORDS. OF CONTENTS
356
DISPATCH MOUSEDOWN EVENT TO CHILD WINDOW
TSubscriptionList:: HitTestFloaters
358
CLOSE VISIBLE REGION
368
END
366
CREATE UNIQUE TOKEN FOR CHILD
YES
CLICK IN CHILD CONTENTS ?
362
NO

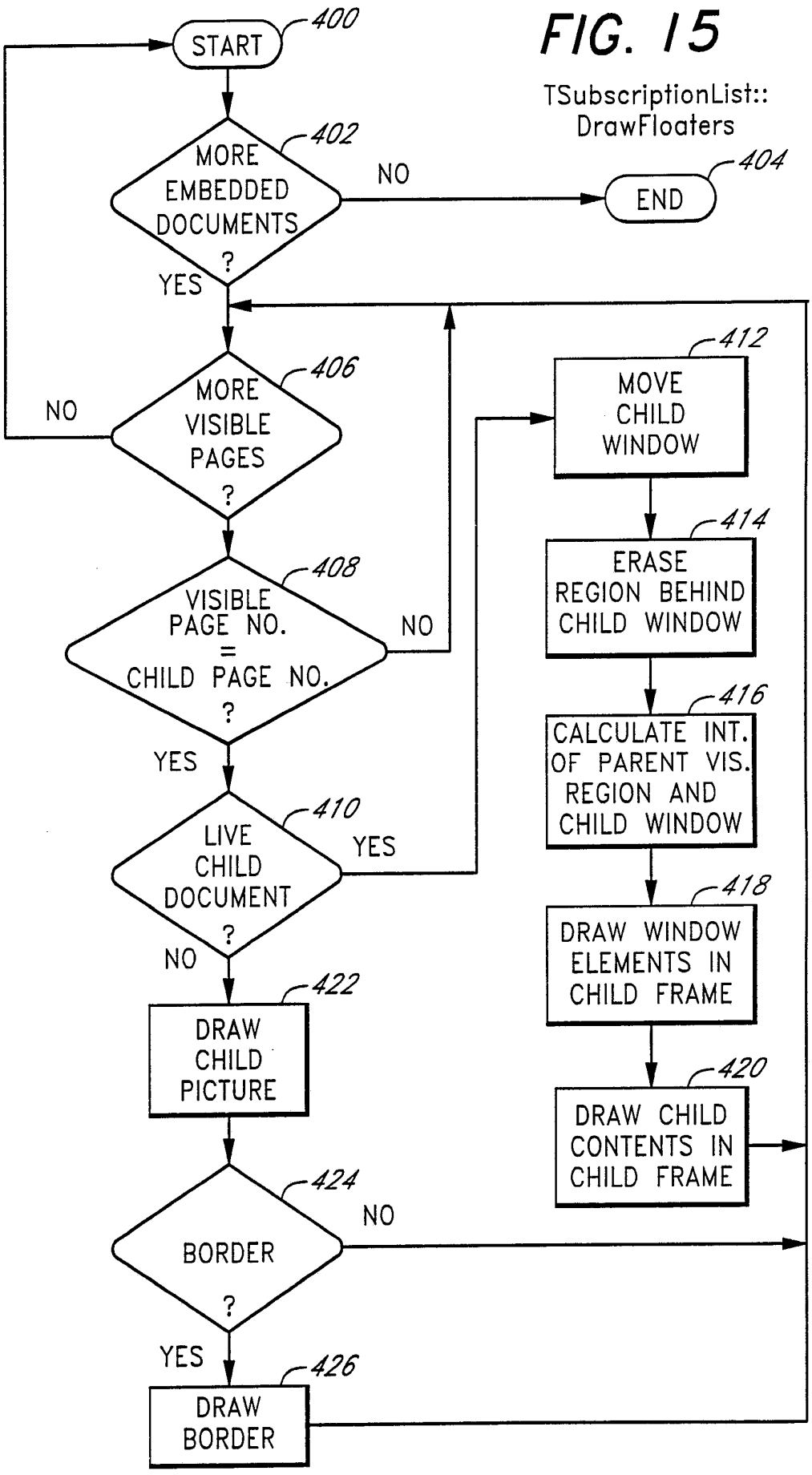



TSubscriptionList::
LaunchPublisherFromToken

FIG. 14

FIG. 15

TSubscriptionList::
DrawFloaters



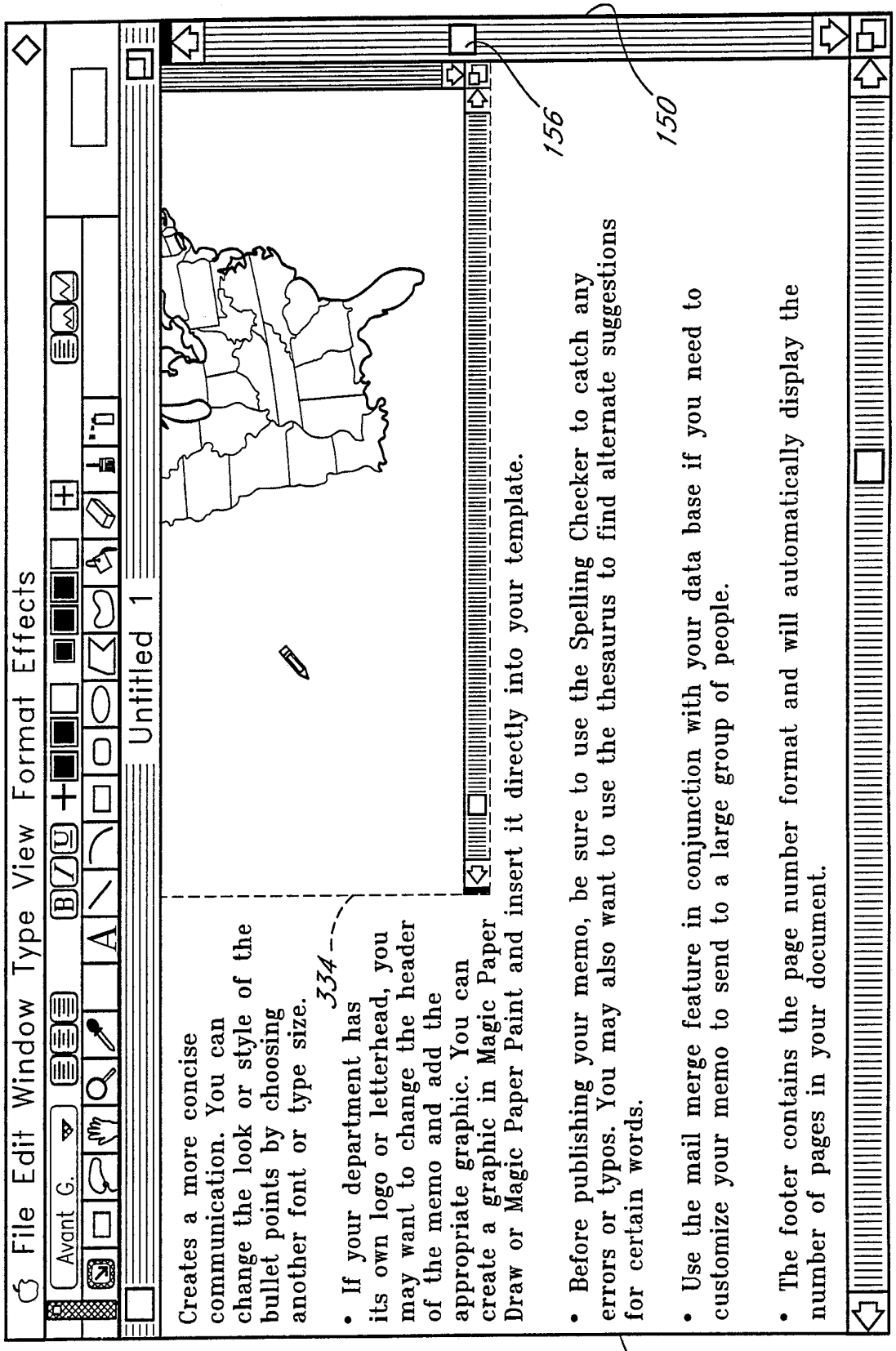
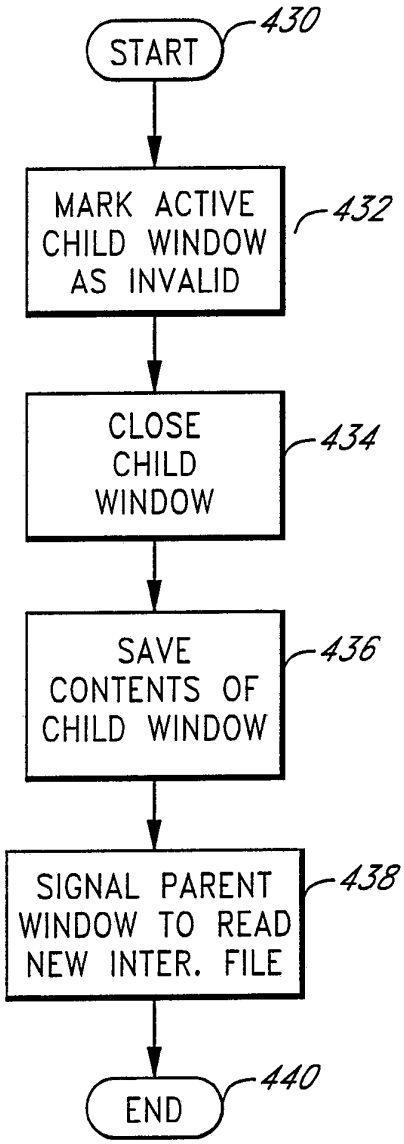

File Edit Window Type View Format Effects
Avant G.
Untitled 1
Creates a more concise communication. You can change the look or style of the bullet points by choosing another font or type size.
• If your department has its own logo or letterhead, you may want to change the header of the memo and add the appropriate graphic. You can create a graphic in Magic Paper Draw or Magic Paper Paint and insert it directly into your template.
• Before publishing your memo, be sure to use the Spelling Checker to catch any errors or typos. You may also want to use the thesaurus to find alternate suggestions for certain words.
• Use the mail merge feature in conjunction with your data base if you need to customize your memo to send to a large group of people.
• The footer contains the page number format and will automatically display the number of pages in your document.
334
156
150
190

FIG. 16

FIG. 17




TDocument::CloseLink

File Edit Window Type View Format Spelling
Avant G.
Avant Garde
7
1 col
Untitled 1
TO: All Employees
From: Memo Department
Subject: Memos
• We have designed a new template for memos and would like all of you to use it for internal correspondence. Consider this a general guideline and make changes as necessary.
We recommend that you use the "bullet point" format whenever possible. This creates a more concise communication. You can change the look or style of the bullet points by choosing another font or type size.
If your department has its own logo or letterhead, you may want to change the header of the memo and add the appropriate graphic. You can create a graphic in Magic Paper Draw or Magic Paper Paint and insert it directly into your template.
Before publishing your memo, be sure to use the Spelling Checker to catch any errors or typos.
193
190

FIG. 18

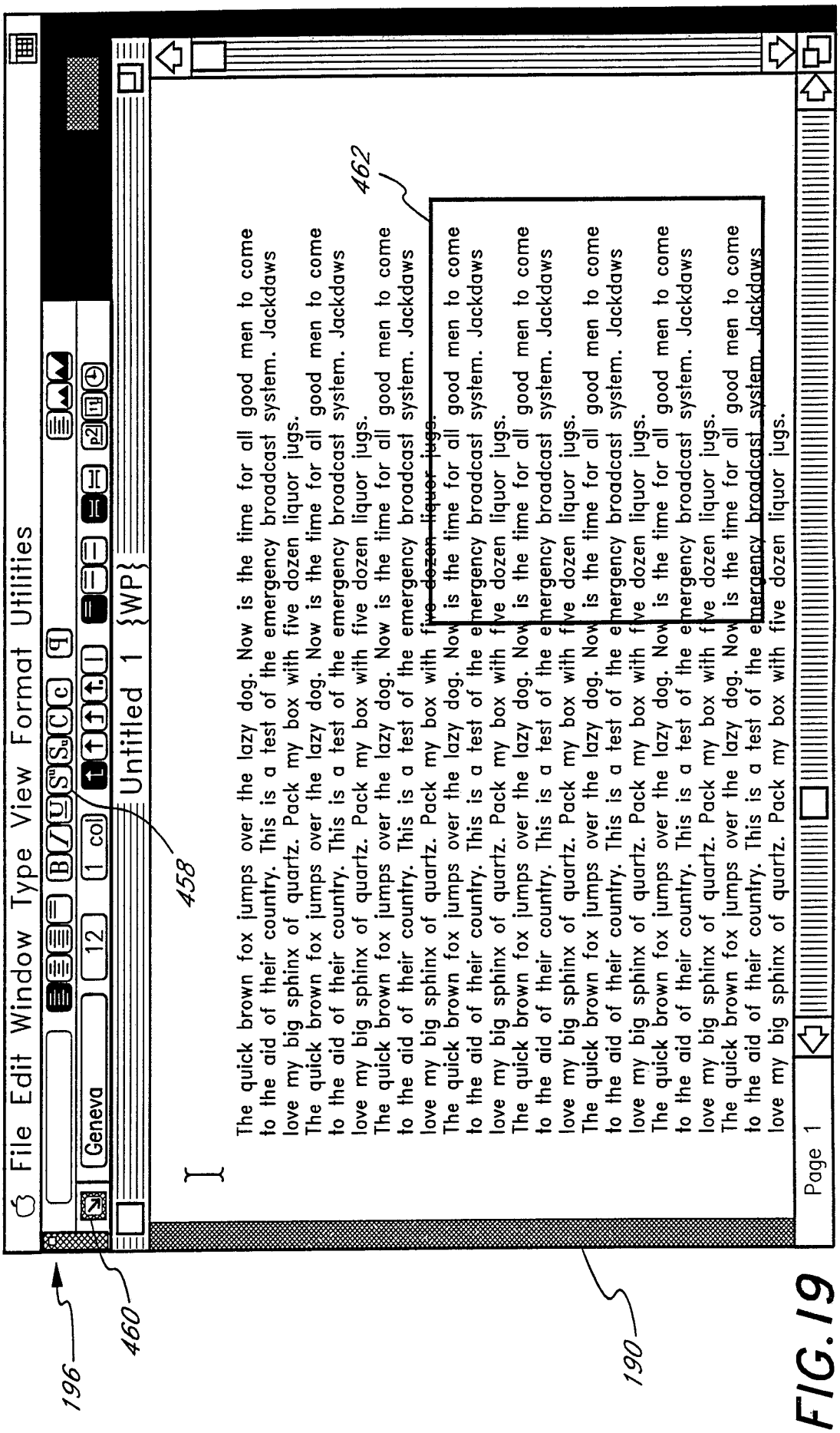

File Edit Window Type View Format Utilities
Geneva
12
1 col
Untitled 1 {WP}
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is a test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
Page 1
458
462
460
196
190

FIG. 19

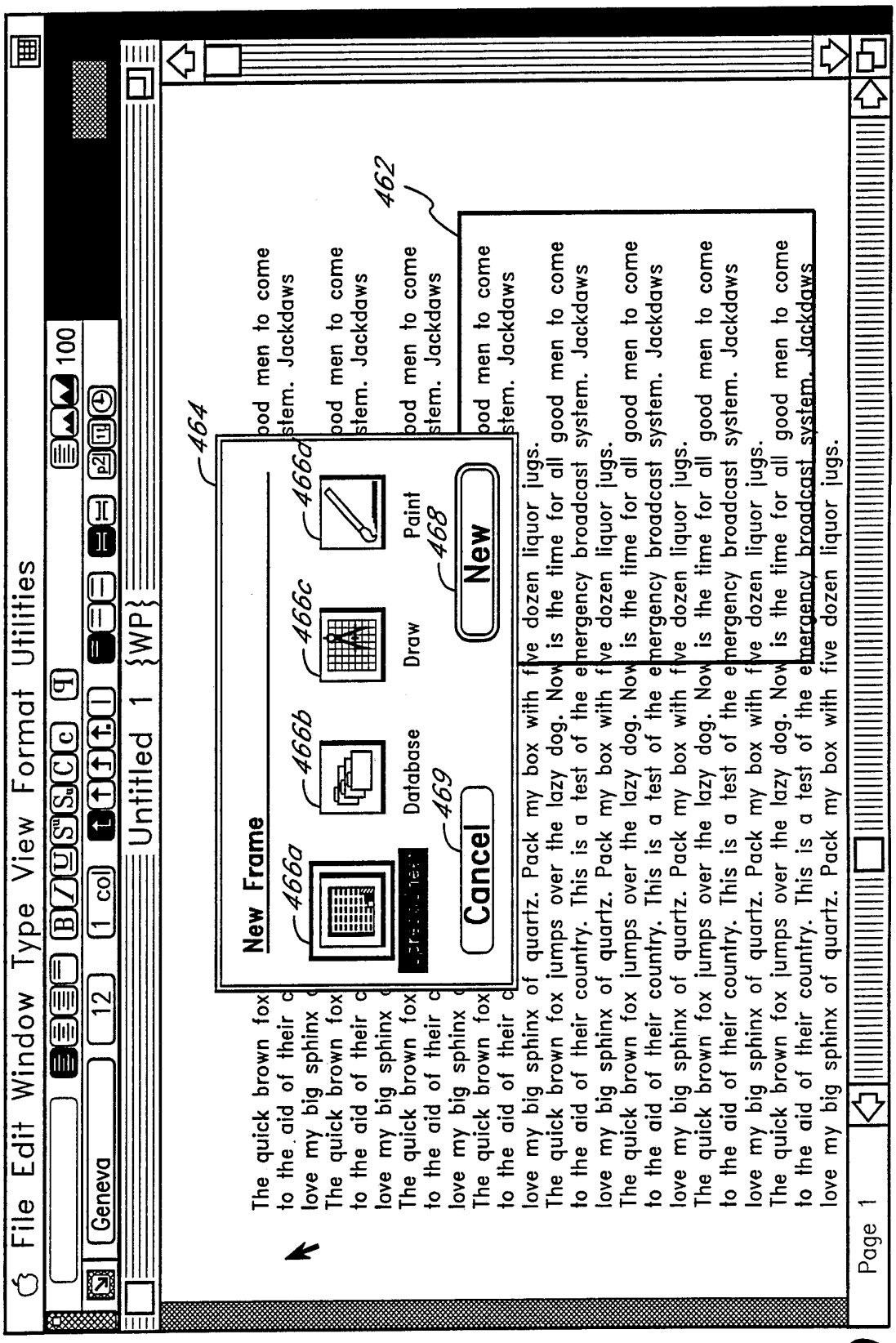

File Edit Window Type View Format Utilities
Geneva
12
1 col
Untitled 1 {WP}
100
New Frame
Database
Draw
Paint
Cancel
New
462
464
466a
466b
466c
466d
468
469
The quick brown fox
to the aid of their c
love my big sphinx
od men to come
stem. Jackdaws
Page 1

FIG. 20

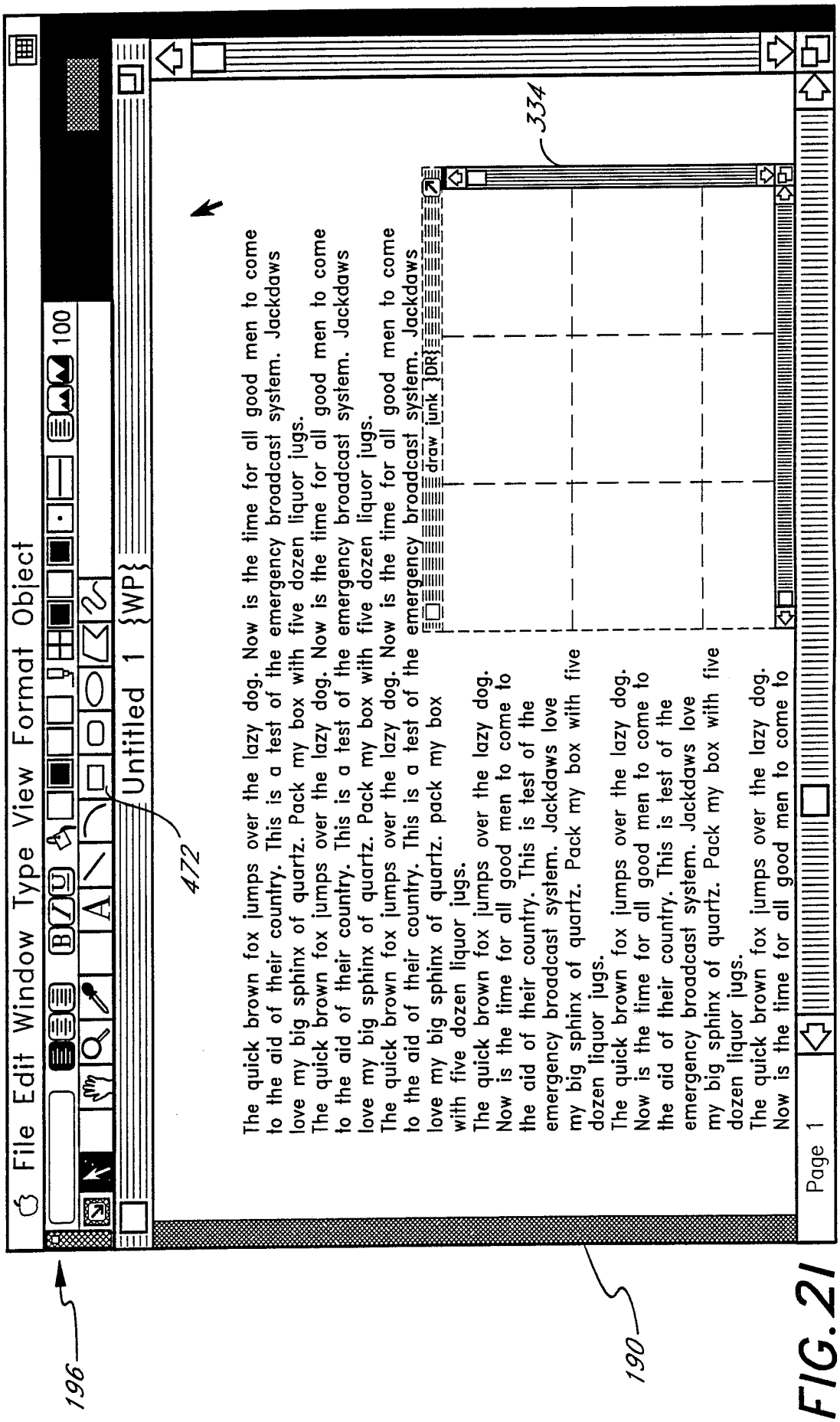
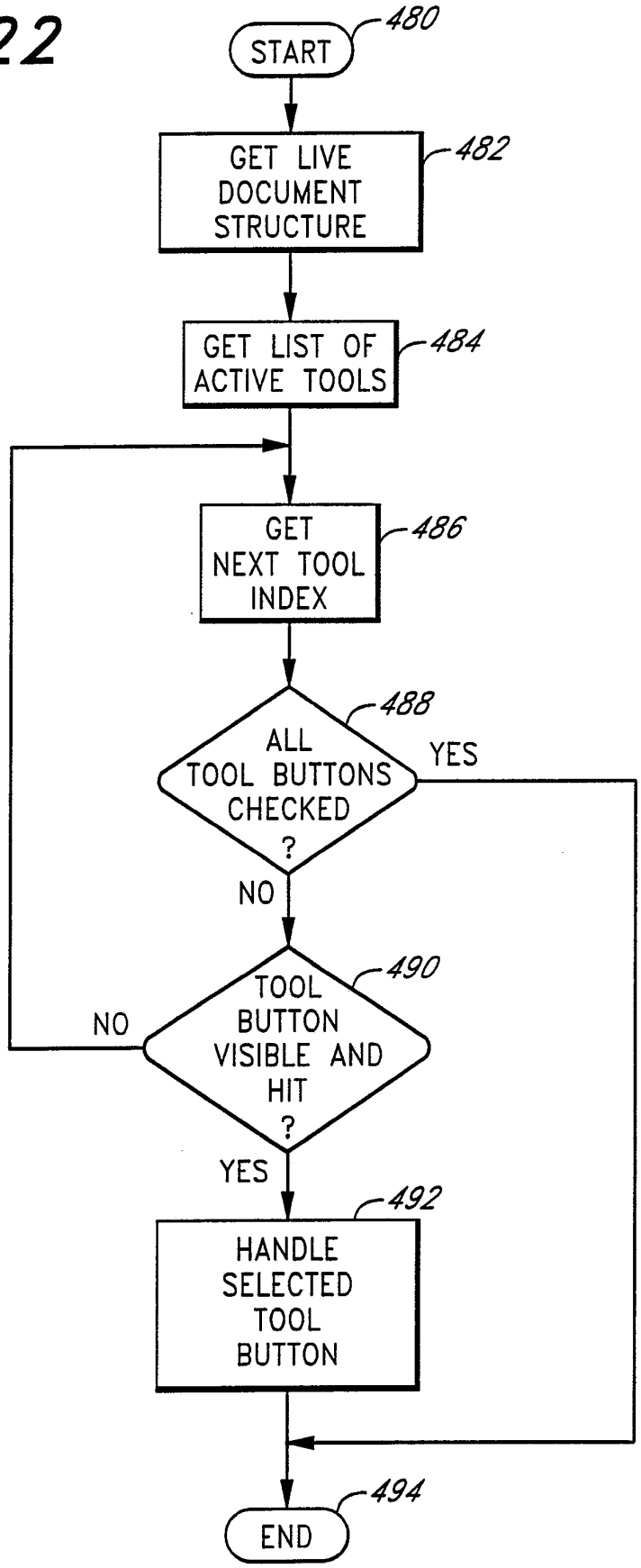

File Edit Window Type View Format Object
Untitled 1 {WP}
100
472
334
draw_junk {DR}
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is a test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
Page 1
196
190

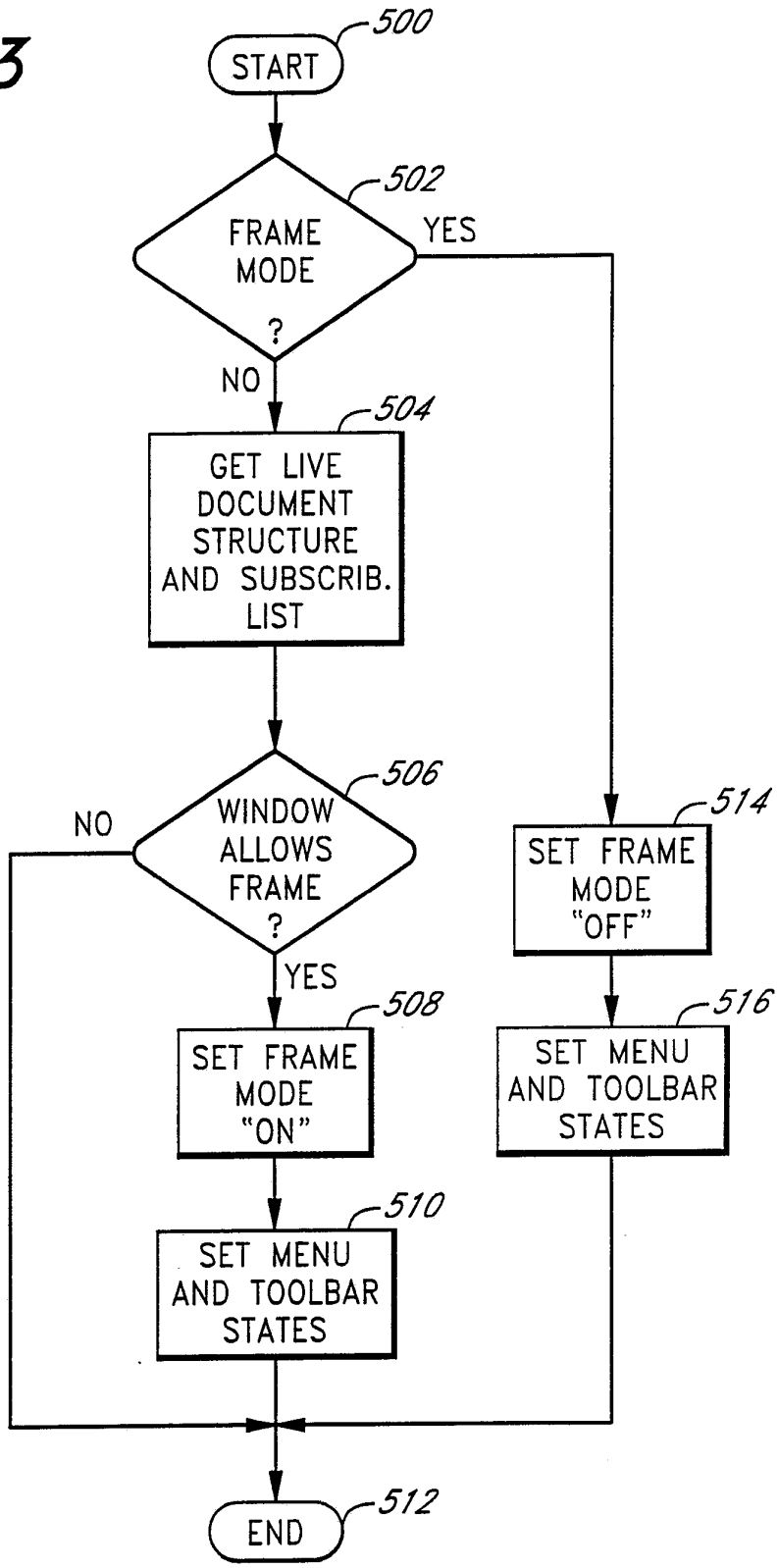
FIG. 21

FIG. 22



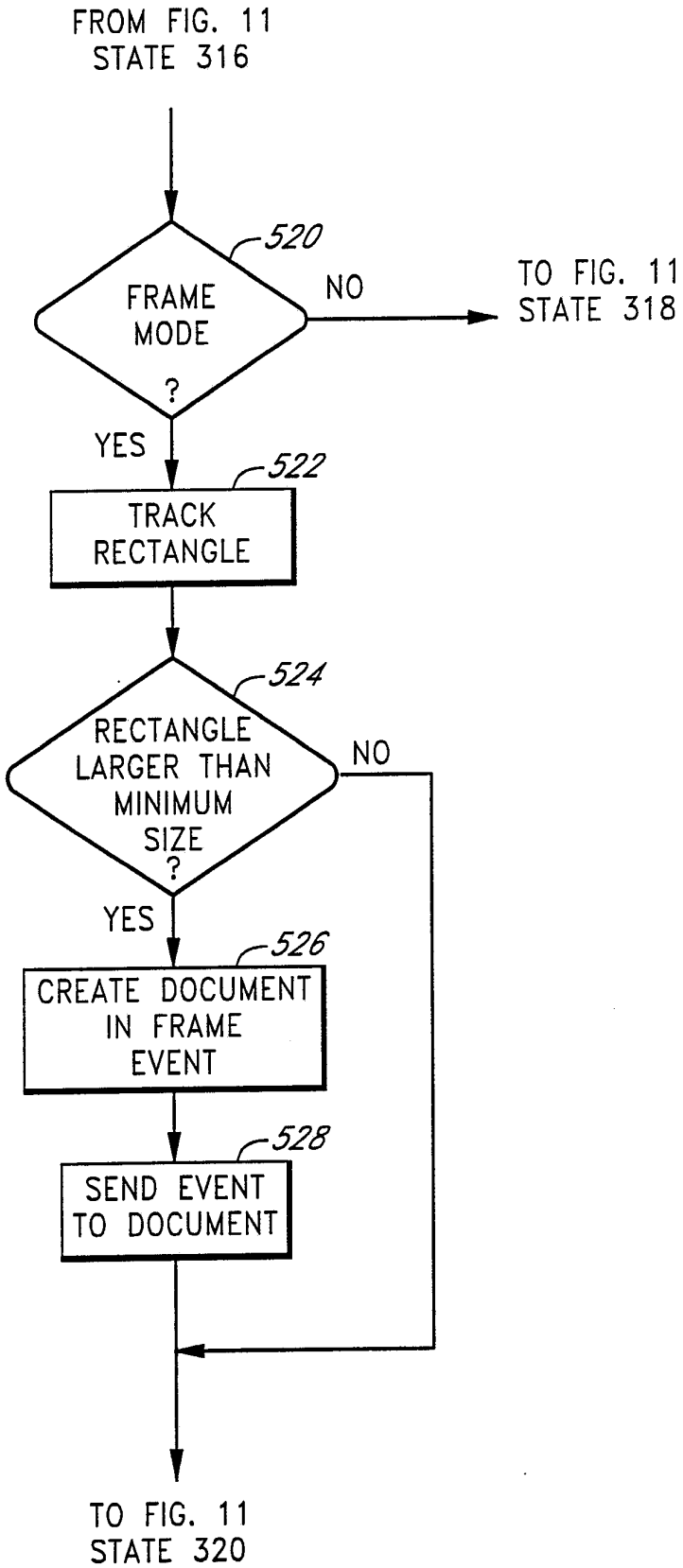

TToolBar::DoMouseDown

FIG. 23



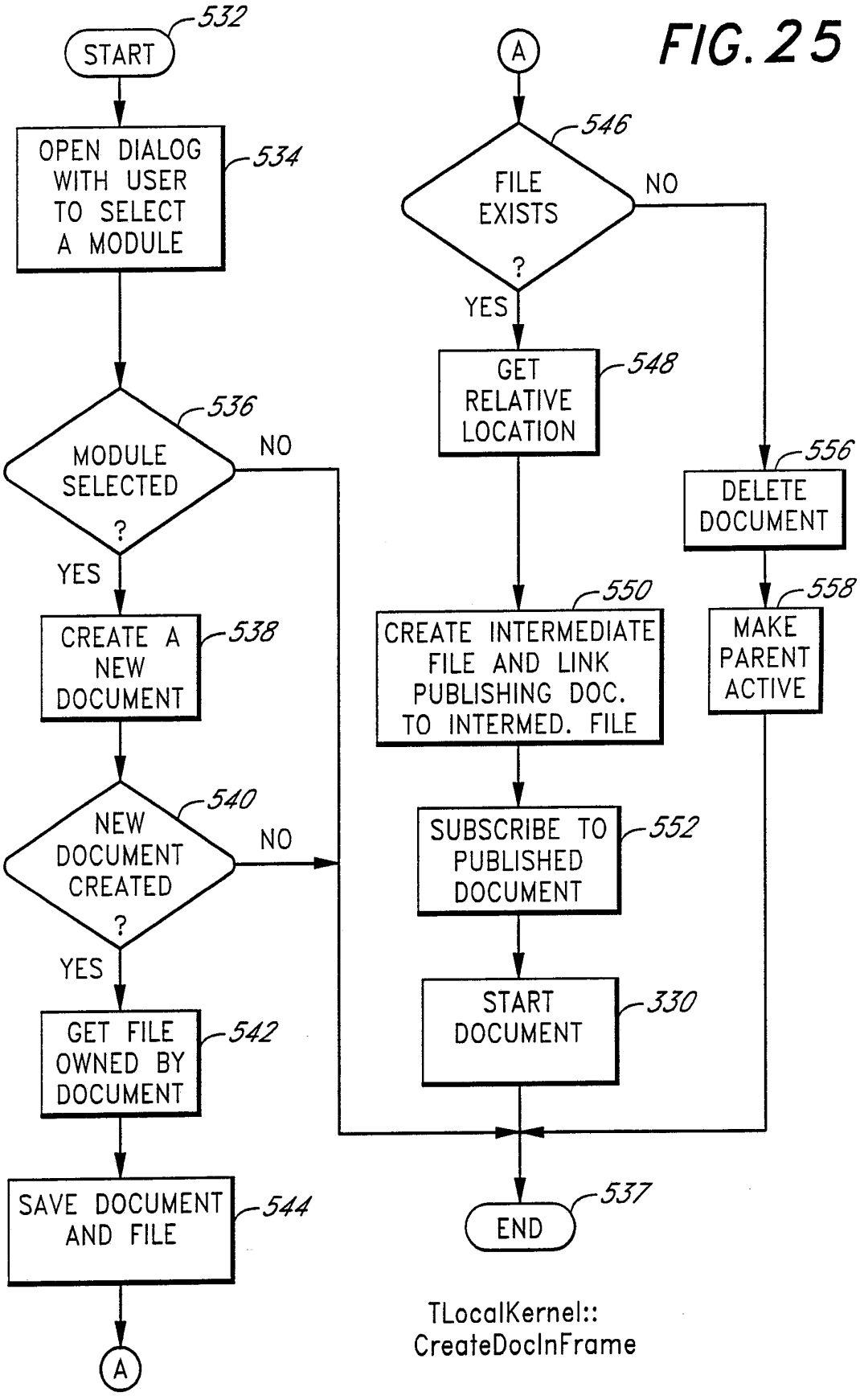

TLocalKernal::
HandleCreateFrame

FIG. 24




TFrame::DoMouseDown(cont.)

FIG. 25




TLocalKernel::
CreateDocInFrame

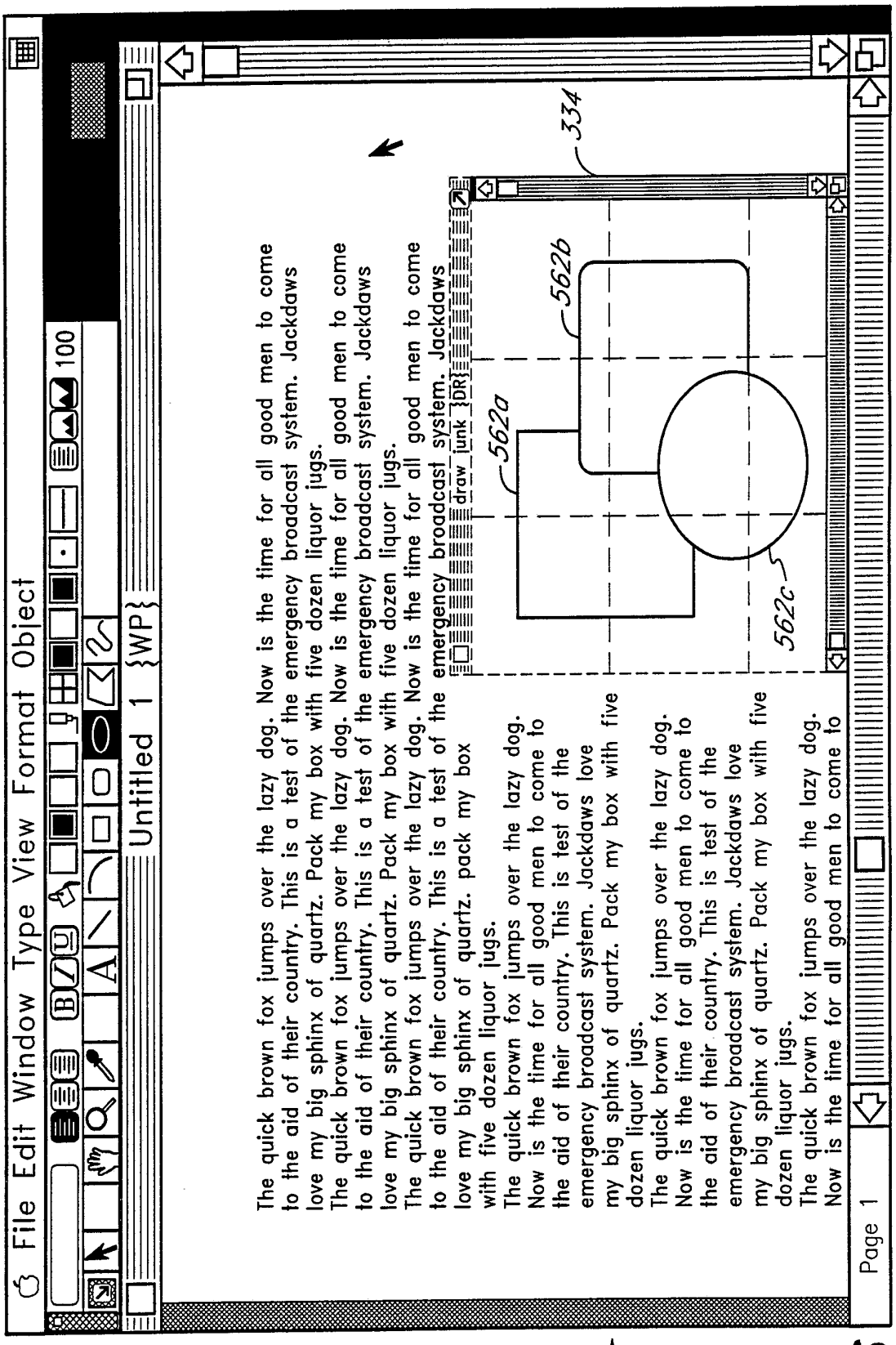

File Edit Window Type View Format Object
Untitled 1 {WP}
100
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is a test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
draw_junk {DR}
334
562b
562a
562c
Page 1
190

FIG. 26

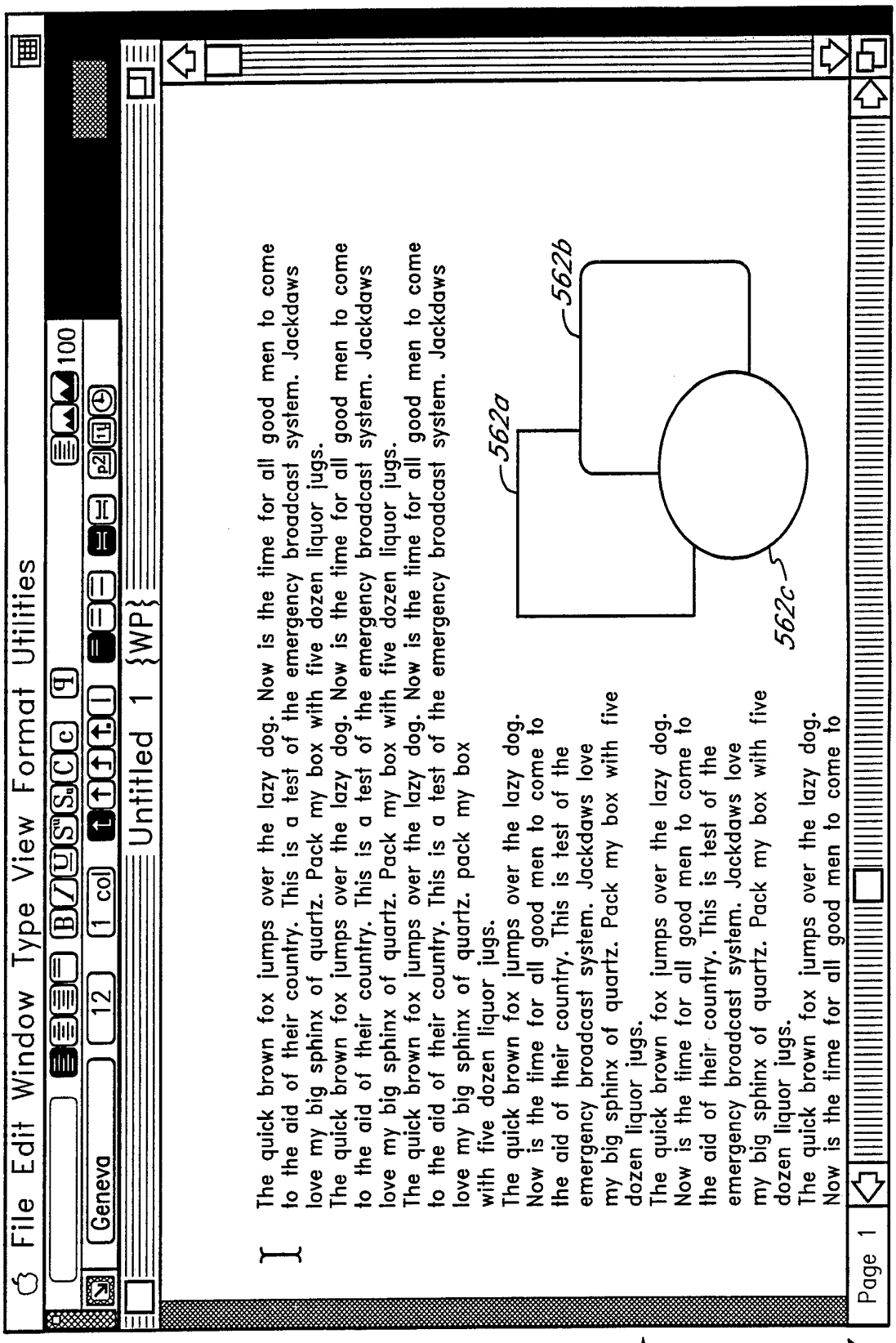

File Edit Window Type View Format Utilities
Geneva
12
1 col
Untitled 1 {WP}
100
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is a test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is a test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is a test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to
562a
562b
562c
Page 1
190

FIG. 27

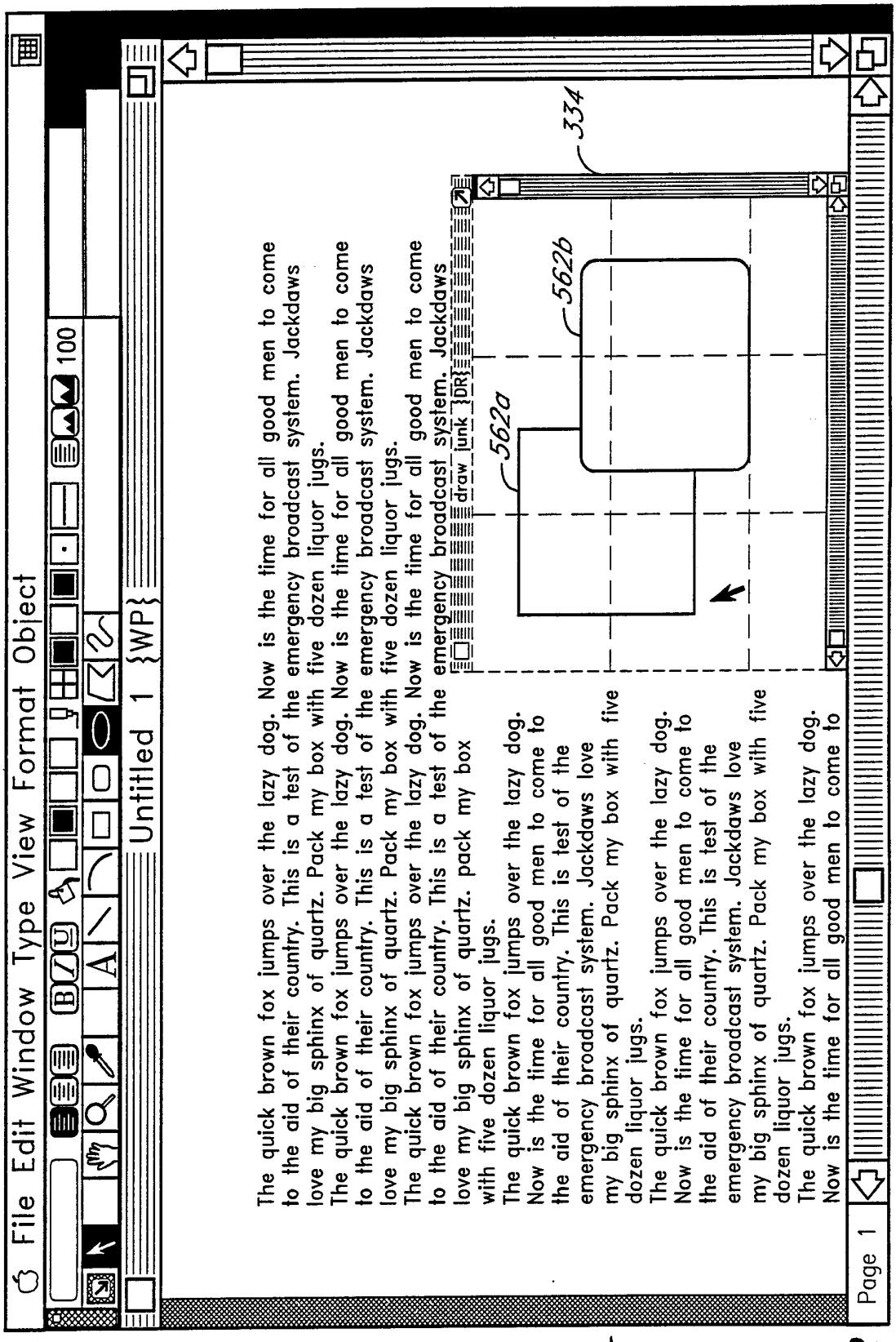

File Edit Window Type View Format Object
Untitled 1 {WP}
100
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is a test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is a test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is a test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to
draw_junk {DR}
562a
562b
334
Page 1
190

FIG. 28

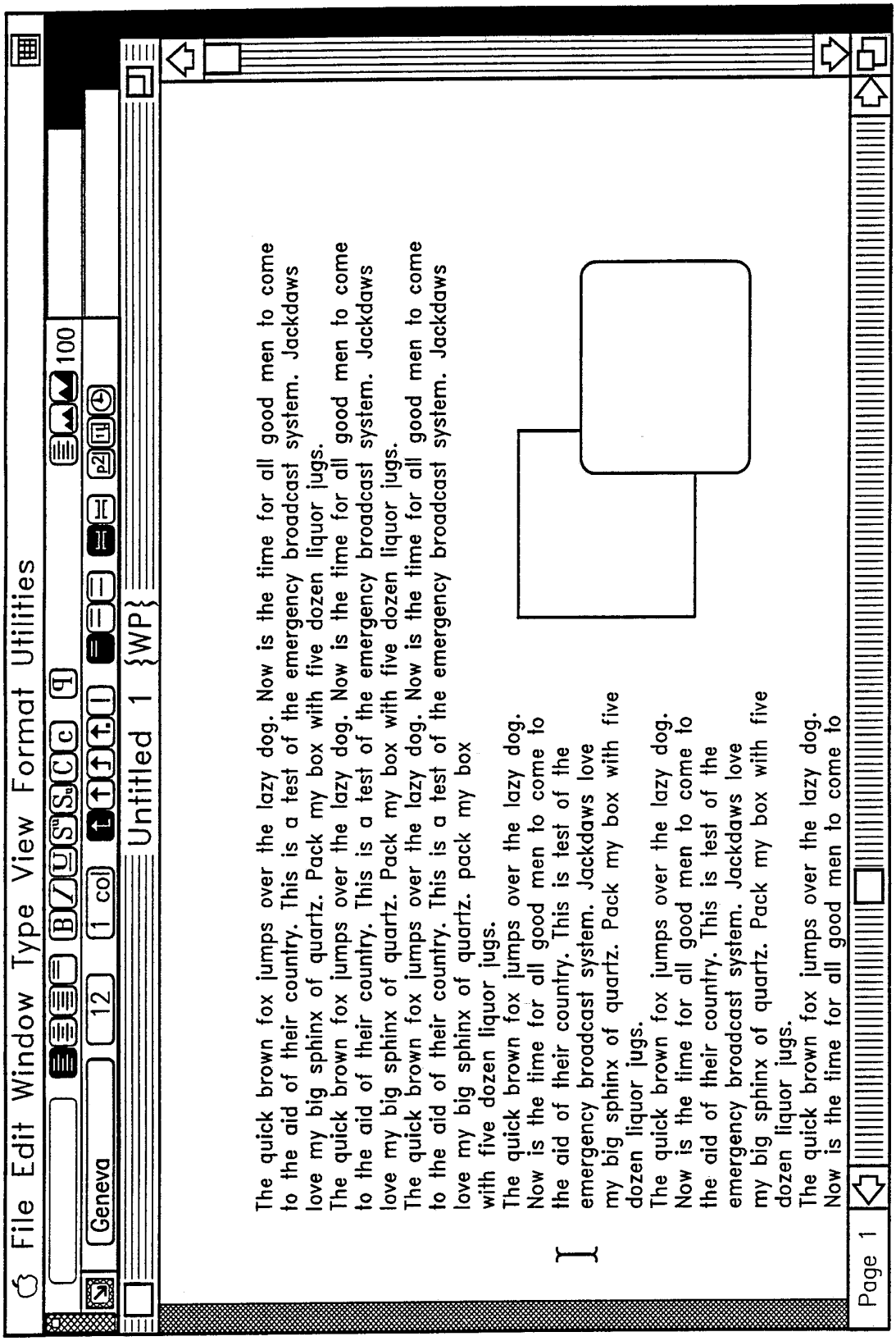

File Edit Window Type View Format Utilities
Geneva
12
1 col
100
Untitled 1 {WP}
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is a test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is a test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is a test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to the aid of their country. This is test of the emergency broadcast system. Jackdaws love my big sphinx of quartz. Pack my box with five dozen liquor jugs.
The quick brown fox jumps over the lazy dog. Now is the time for all good men to come to
Page 1

FIG.29

# INTERNATIONAL SEARCH REPORT

International application No.
PCT/US92/06634

**A. CLASSIFICATION OF SUBJECT MATTER**

IPC(5) :G09G 1/16
US CL :340/721
According to International Patent Classification (IPC) or to both national classification and IPC

**B. FIELDS SEARCHED**

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 340/721, 723, 724; 395/144, 145, 146, 147, 148, 151

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

NONE

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| X<br>Y | US, A, 4,962,475 (HERNADEZ) 09 October 1990, col. 3, line 7 to col. 8, line 59 Fig. 1-5. | 1,2,5,6,9,10,16-18,21-24<br>3,4,7,8,13-15, 19-20 |
| Y,P | US, A, 5,051,930 (KUWABARA ET AL.) 24 September 1991, col. 4, line 45 to col. 16, line 36 Fig. 1-17. | 7,8,13,14,15, 19,20 |
| A | US, A, 4,823,303 (TERASAWA) 18 April 1989, col. 2, line 58 to col. 4, line 24, Fig. 4-5. | 1-10, 13-24 |
| A | US, A, 4,663,615 (HERNANDEZ ET AL.) 05 May 1987, col. 3, line 16 to col. 8, line 59 Fig. 2-9. | 1-10, 13-24 |

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

| | | | |
|---|---|---|---|
| * | Special categories of cited documents: | "T" | later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention |
| "A" | document defining the general state of the art which is not considered to be part of particular relevance | "X" | document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone |
| "E" | earlier document published on or after the international filing date | "Y" | document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art |
| "L" | document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified) | "&" | document member of the same patent family |
| "O" | document referring to an oral disclosure, use, exhibition or other means | | |
| "P" | document published prior to the international filing date but later than the priority date claimed | | |

| | |
|---|---|
| Date of the actual completion of the international search<br>29 SEPTEMBER 1992 | Date of mailing of the international search report<br>15 DEC 1992 |
| Name and mailing address of the ISA/ US<br>Commissioner of Patents and Trademarks<br>Box PCT<br>Washington, D.C. 20231<br>Facsimile No. NOT APPLICABLE | Authorized officer<br>FM GIN GOON<br>NGUYEN NGOC-HO<br>INTERNATIONAL DIVISION<br>Telephone No. (703) 305-4877 |

Form PCT/ISA/210 (second sheet)(July 1992)*

# INTERNATIONAL SEARCH REPORT

International application No.
PCT/US92/06634

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

| Category* | Citation of document, with indication, where appropriate, of the relevant passages | Relevant to claim No. |
|---|---|---|
| A | US, A, 4,975,690 (TORRES) 04 December 1990, col. 3, lines 59 to col. 8, line 65 Fig. 1-3. | 1-10, 13-24 |