



(22) Date de dépôt/Filing Date: 1997/10/17

(41) Mise à la disp. pub./Open to Public Insp.: 1998/06/09

(45) Date de délivrance/Issue Date: 2004/11/30

(30) Priorité/Priority: 1996/12/09 (08/762,237) US

(51) Cl.Int.⁶/Int.Cl.⁶ G06F 3/033, G06F 17/40

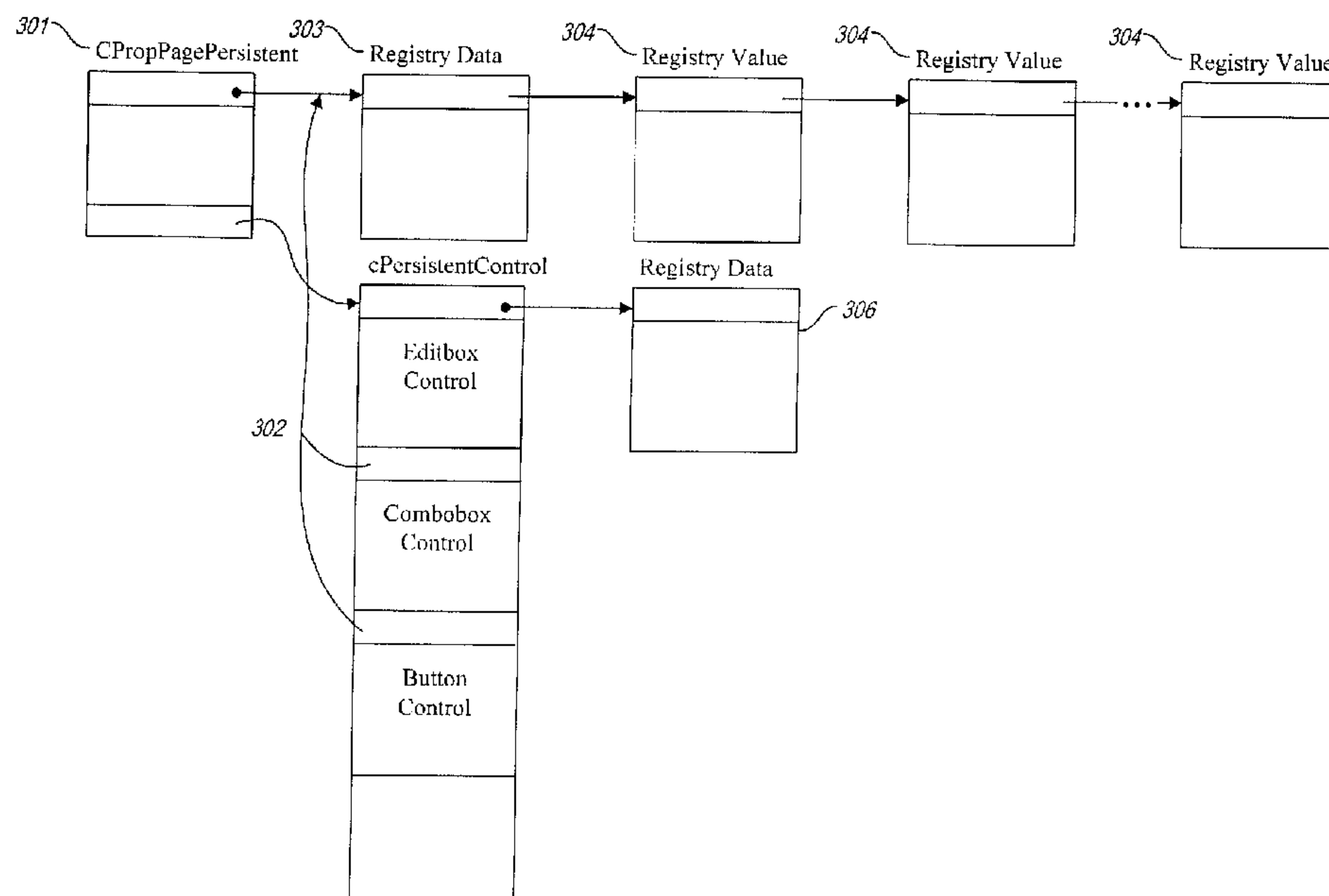
(72) Inventeur/Inventor:
ANDREW, FELIX GTI, US

(73) Propriétaire/Owner:
MICROSOFT CORPORATION, US

(74) Agent: OYEN WIGGS GREEN & MUTALA

(54) Titre : METHODE ET SYSTEME PERMETTANT DE FAIRE AUTOMATIQUEMENT DES ENREGISTREMENTS
PERSISTANTS DE COMMANDES DANS UN ENVIRONNEMENT A FENETRAGE

(54) Title: METHOD AND SYSTEM FOR AUTOMATIC PERSISTENCE OF CONTROLS IN A WINDOWING
ENVIRONMENT



(57) Abrégé/Abstract:

A computer system for persistently storing information entered by a user into an edit control of a property page. In a preferred embodiment, the computer system has a registry for storage of information identified by a key. An application program has a resource file that contains a definition of controls for the property page. The controls include the edit control designated as editable by the user and a buddy control that is adjacent to the edit control, that is designated as not visible, and that has text which specifies a key within the registry. The computer system displays the property page by detecting that the buddy control is adjacent to the edit control and is designated as not visible, by retrieving the text for the buddy control, by retrieving information from the registry using the key specified by the retrieved text, and by initializing the edit control to the retrieved information. The computer system also allows a user to modify the information of the edit control and then stores the modified information of the edit control in the registry using the key specified by the retrieved text to effect persistent storage of the information for the edit control in the registry.



METHOD AND SYSTEM FOR AUTOMATIC PERSISTENCE
OF CONTROLS IN A WINDOWING ENVIRONMENT

Abstract of the Disclosure

A computer system for persistently storing information entered by a user into an edit control of a property page. In a preferred embodiment, the computer system has a registry for storage of information identified by a key. An application program has a resource file that contains a definition of controls for the property page. The controls include the edit control designated as editable by the user and a buddy control that is adjacent to the edit control, that is designated as not visible, and that has text which specifies a key within the registry. The computer system displays the property page by detecting that the buddy control is adjacent to the edit control and is designated as not visible, by retrieving the text for the buddy control, by retrieving information from the registry using the key specified by the retrieved text, and by initializing the edit control to the retrieved information. The computer system also allows a user to modify the information of the edit control and then stores the modified information of the edit control in the registry using the key specified by the retrieved text to effect persistent storage of the information for the edit control in the registry.

DescriptionMETHOD AND SYSTEM FOR AUTOMATIC PERSISTENCE
OF CONTROLS IN A WINDOWING ENVIRONMENT

5

Technical Field

The present invention relates generally to a computer system for storing of data and, in particular, a system for persistently storing data of a control of a window.

10

Background of the Invention

Many computer systems provide a window-based user interface through which a user can interface with a computer program. A window is an area, typically rectangular, of a display device in which the computer program displays its output. A window may contain several different types of "fields" into which the computer program can display data. For example, in one field the computer program can display a label to be viewed by a user, and in another field the computer program can echo the characters entered by a user with a keyboard. Each of these fields are typically implemented as a child window of the window in which they are contained. These child windows are also known as "controls."

Figure 1a illustrates a window that contains controls. Because this window contains information relating to properties of a document, it is known as a property sheet. A property specifies certain characteristics of an object, such as a document. For example, the properties of a document may include name of the author and date of creation, whereas the properties of a character may include font and color. The property sheet 101 contains five property pages: general 102, summary 103, statistics 104, contents 105, and custom 106. Each property page is a child window of the property sheet and contains various controls, which

are child windows of the property page. For example, the custom property page contains combobox 107, edit box 108, check box 109.

Certain windowing environments, such as Windows® by Microsoft Corporation, provide various tools to help a programmer generate programs that use windows and controls. In particular, to create a property page, such as the custom property page, a programmer uses a resource editor to define the layout and characteristics of the property page. These characteristics are stored in a resource file, which is opened and used by the computer program. Table 1 contains an excerpt of a resource file corresponding to the custom property page. This portion of the resource file contains the definition of the combobox 107, the edit box 108, the check box 109, and their associated labels. Each line contains the definition of a control. Line 1 corresponds to the label “Type:” to the left of combobox 107 and indicates that the control is a text control (*i.e.*, “LTEXT”) and that the control is static and located at the specified location. The location represented by “[location]” in each line indicates the upper left coordinates and the lower right coordinates of the location of the control within the custom property page. Line 2 corresponds to the combobox 107, indicates that the control is a combobox (*i.e.*, “COMBOBOX”), and contains a unique identifier for the combobox (*i.e.*, “IDC_COMBO1”). Line 2 also indicates that the combobox has a drop down list and that when a user tabs through the controls, the tab will stop at this combobox. Line 3 corresponds to the label “Value:” and indicates that the control is a text control and is static. Line 4 corresponds to the edit box 108, indicates that the control is an edit box (*i.e.*, “EDITTEXT”), and contains a unique identifier for the control. Line 4 also indicates that the control will scroll horizontally and that the tab will stop at this control. Line 5 corresponds to the check box 108, indicates that the control is a control (*i.e.*, “CONTROL”), and contains a unique identifier for the control. Line 5 also

contains a label for the control (*i.e.*, “Link To Content”) and indicates that the control is a check box and that the tab will stop at this control.

Table 1

5	BEGIN	.	
		.	
		.	
1	LTEXT	“Type:”, IDC_STATIC, [location]	
10 2	COMBOBOX	IDC_COMBO1, [location], CBS_DROPDOWNLIST, WS_TABSTOP	
3	LTEXT	“Value:”, IDC_STATIC, [location],	
4	EDITTEXT	IDC_EDIT1, [location], ES_AUTOHSCROLL, WS_TABSTOP	
15 5	CONTROL	“ <u>L</u> ink To Content”, IDC_CHECK1, “Button”, [location]	
		BS_AUTOCHECKBOX, WS_TABSTOP	
		.	
		.	
20	END		

The use of resource files allows for various changes to be made to the arrangement and content of the controls without the need to modify the computer program that displays the controls. For example, the controls corresponding to the labels of the various boxes can be changed by simply modifying the resource file. The computer program would then be re-built (but not re-compiled). The next time the re-built computer program is executed, the program will open the resource file with the modifications and will display the modified controls.

Figures 1b-1d illustrate the operation of various types of controls. Figure 1b illustrates the operation of a combobox. The combobox 107 comprises an edit box 107a and a list box 107b. When the user selects the arrow to the right of the edit box 107a, the list box 107b is displayed. The list box contains a list of the possible values that can be entered into the edit box 107a. A user can select one of the possible values from the list box 107b by selecting a value with a mouse and mouse pointer or can use the keyboard to enter a value directly into

the edit box 107a. Figure 1c illustrates the operation of an edit box. The user can type in any information into an edit box, subject to editing performed by the underlying computer program. Figure 1d illustrates the operation of a check box. The check box 109 can be checked or unchecked by a user using a mouse and
5 mouse pointer.

Typical computer programs allow for persistent storage of data entered into the various controls of a property page. Such data is typically persistently stored in a file on disk. Each computer program controls the retrieving of the values from the various controls and the storing of the values
10 within the file. Thus, each developer of a computer program that persistently stores the values of the controls needs to develop code that implements such storing.

Typical computer programs use hard-coded values (*i.e.*, defined in the source code) for the various list boxes, including for the list box of a
15 combobox. Thus, to change the content of a list box, a developer would change and recompile the source code. Such changes of the source code are error prone and costly.

Summary of the Invention

20 The present invention provides a control method and system for altering the values of controls displayed within a window. The control system has resource information that contains a definition of each control within the window. For each control, the control system determines whether a buddy control is associated with the control. The buddy control has buddy control
25 information and preferably has no visible manifestation when the window is displayed. When the control has a buddy control, the control system retrieves the buddy control information and displays the retrieved information as part of the control. A user can then modify the definition of the buddy control by modifying

the buddy control information so that when the control is subsequently displayed, the modified buddy control information is displayed. The modified buddy control information is preferably displayed without modifying the computer program that controls the displaying of the window. Also, the buddy control is
5 preferably adjacent to the control. Moreover, when the control is a list box, the buddy control information is a list of items to be displayed in the list box.

In another aspect the control system provides a general mechanism for passing parameter data to a computer program to affect a behavior of the computer program. The computer program has a window and resource
10 information defining controls within the window. The control system defines a parameter control within the resource information in which the parameter control has no visible affect on the display of the window and contains parameter data to be passed to the computer program. The computer program retrieves the definition of the parameter control from the resource information, extracts the
15 parameter data from the retrieved definition, and performs the behavior indicated by the parameter data so that the behavior of the computer program can be changed by changing the definition of the control. The parameter data preferably contains a key for persistently storing a value for a control or contains information to be displayed when a control is displayed, such as a list of items to
20 be displayed in a list box.

Brief Description of the Drawings

Figure 1a illustrates a window that contains controls.

Figure 1b illustrates the operation of a combobox.

25 Figure 1c illustrates the operation of an edit box.

Figure 1d illustrates the operation of a check box.

Figure 2 is a block diagram illustrating a computer system upon which the present invention may be practiced.

Figure 3 is a block diagram illustrating various data structures used in a preferred embodiment.

Figure 4 is a flow diagram of the method BaseDlgProc of the CPropPagePersistent class.

5 Figure 5 is a flow diagram of the method DoCommand of the CPropPagePersistent class.

Figure 6 is a flow diagram of the method SetActive of the CPropPagePersistent class.

10 Figure 7 is a flow diagram of the method Display of the CPropPagePersistent class.

Figure 8 is a flow diagram of the method Apply of the CPropPagePersistent class.

Figure 9 is a flow diagram of the method Load of the CPersistentEditControl class.

15 Figure 10 is a flow diagram of the method GetAndParseBuddy of the CPersistentEditControl class.

Figure 11 is a flow diagram of the method Display of the CPersistentEditControl class.

20 Figure 12 is a flow diagram of the method Load of the CPersistentComboControl class.

Figure 13 is a flow diagram of the method Display of the CPersistentComboControl class.

Figure 14 is a flow diagram of the method Load of the CPersistentButtonControl class.

25 Figure 15 is a flow diagram of the method Display of the CPersistentButtonControl class.

Detailed Description of the Invention

The present invention provides a control method and system for persistently storing user-entered values for controls and for modifying various controls. In a preferred embodiment, the control system defines a window class and, for each type (*e.g.*, combobox, text box) of control, a control class. These classes provide functionality for persistently storing control values and for modifying the content of controls. A computer program that uses the control system instantiates a window object to manage the window that contains the controls, and the window object instantiates a control object for each control to manage the control. When the window is first displayed, the window object requests each control object to load the control value from the persistent storage. When user indicates to apply the current control values (*e.g.*, clicking an "OK" button), the window object requests each control object to store the control value in the persistent storage. Thus, the control values are persistently stored and then are retrieved when the window is displayed. The persistent storage is preferably the system registry provided by the Microsoft Windows® operating system. However, other persistent storage such as a data base or a file may be used.

To persistently store the control values, the control system preferably assigns a unique key to each control and uses that key to store and retrieve the control value. In one embodiment, the unique key can include a combination of program identifier, window identifier, and control identifier. Thus, the control system generates such a unique key when the value for a control is stored and regenerates the same key when the value is retrieved. The generation of such a combination, however, means that the control system would not be able to retrieve the value stored for a control that is moved from one window to another. In particular, before the control is moved, the control system would generate a key based on the identifier of the current window. After the control is moved, the system would generate a new key based on the identifier of

the new window. However, with this new key, the value stored with the other key cannot be retrieved. For example, the control system may assign a key of "Custom\IDC_COMBOBOX1" to the combobox 107 of Figure 1a because its window has the title "Custom" and the combobox has an identifier of "IDC_COMBOBOX1." Thus, the value of the combobox would be stored with that key. However, if the combobox were moved to the window with the title "General," then when the general window is displayed, the key "General\IDC_COMBOBOX1" would be generated. Using this key, the control system would try to retrieve the persistently stored value for the combobox. However, since the control was moved, the key is incorrect and the persistently stored value would not be retrieved.

To address this problem of incorrect key generation, the control system provides a mechanism which allows each control to be assigned a key that is independent of the window in which the control is contained. With such a window-independent key, a control can be moved to a different window and the value that was persistently stored before the control was moved can still be retrieved. The mechanism allows the window-independent key to be stored in the resource file along with the definition of the control. In one embodiment, the definition of a control can be expanded to include an indication of the key for the control. For example, a parameter could be added to a line that defines a control to specify the window-independent key. Such an expansion of the definitions could be accomplished by performing a re-specification of the format of the resource file.

In a preferred embodiment, a window-independent key is specified without a re-specification of the format of the resource file. The control system uses a "buddy control" associated with each control that contains the window-independent key as its text (*e.g.*, the unique identifier in the resource file). The buddy control for a specified control is defined in the resource file as a static and

a not visible text control that immediately follows the specified control. Thus, the control object for the specified control can determine if a buddy control is defined by retrieving the next control and determining whether it is a static and a not visible text control. If a buddy control is defined, then the text of that buddy
5 control specifies the key that is to be used when storing and retrieving the value for the specified control. Moreover, since the buddy control is not visible, it does not affect the appearance of the window.

In addition to specifying the key, the presence or absence of buddy control can be used as a flag to indicate whether the value of a control should be
10 persistently stored. That is, if no buddy control is defined for a certain control, then the value for that control would not be stored persistently. Conversely, if a buddy control is defined, then the value would be persistently stored using the specified key. In general, the use of a buddy control can be used as a mechanism to provide information to the computer program about the corresponding control.
15 Moreover, the use of a control that does not affect the appearance of a window provides a generic mechanism for providing arbitrary information to a computer program.

In certain situations, the assignment of a unique key to each control may result in an unacceptable overhead. For example, the accessing of the
20 registry may involve a disk access for each key. Thus, a preferred control system allows values for certain types of controls to be combined and stored with a single key. In one embodiment, the values for all the button controls of a window are combined and stored under a single key. However, when such values are combined, the corresponding controls cannot be readily moved to a
25 different window. Thus, the decision to implement such combining of values would entail considerations of the savings in access versus the loss in the ability to easily move controls to a different window. In a preferred embodiment, the control system can be implemented to combine all values for all controls of

certain types for all windows of the computer program and to store the combined values with a single key. In this way, the control could be moved to different windows and their persistently stored values could be retrieved.

The control system also allows for the items in a list box to be
5 specified without hard-coding the item into the computer program. In particular, the items of a list box can be stored as the text of a buddy control. Thus, a control object for a list box can determine if a buddy control is defined. If one is defined, then the control object can fill the list box with the items indicated in the text.

10 Table 2 illustrates the use of buddy controls defined in a resource file. As illustrated in this example, the control system automatically stores all control values for comboboxes and check boxes persistently. The control system combines each of these control values and stores them with a single key. The control system uses the presence or absence of a buddy control for an edit box as
15 a flag to indicate whether the value for the edit box should be persistently stored. If a buddy control is present, then its text contains the key for use in storing and retrieving the control value for the edit box. Also, if a combobox has a buddy control, the control system interprets the text of the buddy control as containing the items for the list box of the combobox. Line 2a defines a buddy control for
20 the combobox defined in line 2. The text of this buddy control contains a list of the items for the list box of the combobox. Line 4a defines a buddy control for the edit box defined in line 4. The text of this buddy control contains the key for use when storing and retrieving the value of the edit box.

Table 2

BEGIN			
	.		
	.		
5	.		
1	LTEXT	"Type:", IDC_STATIC, [location]	
2	COMBOBOX	IDC_COMBO1, [location], CBS_DROPDOWNLIST, WS_TABSTOP	
2a	LTEXT	"Text\nDate\nNumber\nYes or No", IDC_STATIC, [location], Not WS_VISIBLE WS_DISABLED	
10			
3	LTEXT	"Value:", IDC_STATIC, [location],	
4	EDITTEXT	IDC_EDIT1, [location], ES_AUTOHSCROLL, WS_TABSTOP	
4a	LTEXT	"HKLM, System\\CurrentControlSet\\Services\\VxD\\VNetSup, ComputerName, 15", IDC_STATIC, [location], Not WS_VISIBLE WS_DISABLED	
15			
5	CONTROL	"Link To Content", IDC_CHECK1, "Button", BS_AUTOCHECKBOX, WS_TABSTOP	
	.		
20	.		
	.		
END			

Figure 2 is a block diagram illustrating a computer system upon which the present invention may be practiced. The computer system 201 comprises a central processing unit, memory, and storage devices, such as a disk drive. The computer system also includes an operating system 202, a registry 203, an application program 204, and a display 205. The control system is typically stored in a computer-readable medium, such as a disk or read-only memory, from which it is loaded into memory for execution. The operating system coordinates the overall operation of the computer system and acts as an interface between the application program and the various devices, such as the display. The registry provides a centralized mechanism through which various application programs can store information persistently. In a preferred embodiment, the information in a registry is hierarchically organized based on a path name (*i.e.*, key) to the data. The registry could be implemented using a database or by storing the information in a file. The application program controls the displaying of various windows. For each window, such as a property page,

the application program instantiates a window object 210 having functionality for managing display of the window. The application program also includes a control object corresponding to each control on the window for managing the control. The present invention is preferably implemented as various C++ class,
5 which define the window and control objects.

Figure 3 is a block diagram illustrating various data structures used in a preferred embodiment. In the following, the invention is described based upon an implementation of the window as a property page. Windowing systems, such as Windows, define standard window and control classes to control the
10 standard operation of the windows and controls, and in particular, a property page and its controls. In a preferred embodiment, the window and control classes of the control system inherit these defined classes and add functionality to implement the various aspects of the invention. In one embodiment, each property page has a corresponding CPropPagePersistent object 301 in memory,
15 which is an instance of the CPropPagePersistent class that inherits the standard property page class. The CPropPagePersistent object contains a pointer to a list of CPersistentControl objects. Each control within the property page has a corresponding CPersistentControl object which is an instance of the CPersistentControl class that inherits the standard control class. The
20 CPersistentControl class is subclassed based on control type: edit box (CPersistentEditControl), combobox (CPersistentComboControl), and button (CPersistentButtonControl). That is, one subclass corresponds to an edit box, another subclass corresponds to a combobox, and the other subclass corresponds to a button. The CPropPagePersistent object also contains a pointer to registry
25 data 303. The registry data 303 contains information to be stored in the registry at a single key. In this example, the values for each combobox and button on the property page are combined and stored with a single key. Thus, all the combobox and button values for a single property page are stored with one key.

When a CPropPagePersistent object directs a CPersistentComboControl or CPersistentButtonControl object to persistently store its value, the object stores its value in a registry value structure 304. After the CPropPagePersistent object directs all the control objects to persistently store their values, the
5 CPropPagePersistent object combines and persistently stores all the control values in the registry value structure 304. Each CPersistentEditControl object contains a pointer to registry data 306. The registry data 306 contains the unique key that was retrieved from the buddy control, if present, for the corresponding CPersistentEditControl object.

10 Figures 4-8 illustrate flow diagrams for various methods of the CPropPagePersistent class. The CPropPagePersistent class has a method BaseDlgProc for processing window messages sent to the property page window, a method DoCommand for indicating that the value of a control has changed, a method SetActive for initializing the values of the controls to the persistently
15 stored values, a method Display for displaying the property page, and a method Apply for persistently storing the values of the controls. The CPropPagePersistent class contains a data member indicating whether a value of a control has changed and thus indicating whether the values need to be persistently stored.

20 Figure 4 is a flow diagram of the method BaseDlgProc of the CPropPagePersistent class. The method serves as a window procedure for the property page window. The method is passed various messages and processes the messages. The method receives an initialization message ("WM_INIT") indicating that the property page should be initialized, a command message
25 ("WM_COMMAND") indicating that a value of a control has been changed, and a notify message ("WM_NOTIFY") indicating whether the property page is being displayed for the first time ("SETACTIVE"), whether the OK button has been selected ("APPLY"), or whether the cancel button has been selected

(“QUERYCANCEL”). The implementation of this method is preferably inherited from the standard property page class. In step 401, if the message is an initialize message, the method continues at step 402, else the method continues at step 404. In steps 404-406, the method stores the passed handle for the window
5 associated for this object and clears the changed and initialization flags. This passed handle is stored so that the other methods of the CPropPagePersistent object can identify the corresponding property page window. In step 402, if the message is a command message, then the method continues at step 407, else the method continues at step 409. In step 407, the method calls the method
10 DoCommand of this CPropPagePersistent object and returns. In step 409, if the message is a notify message, then the method continues at step 410, else the method returns. In step 410, if the notify is an APPLY message, then the method continues at step 411. If the message is a SETACTIVE message, then the method continues at step 414. If the message is a QUERYCANCEL message,
15 then the method continues at step 416. In step 411, if a control value of the property page has been changed, then the changes are to be persistently stored and the method continues at step 412, else the method returns. In step 412, the method invokes the method Apply of this CPropPagePersistent object to store the control values in the registry. In step 413, the method clears the changed flag to
20 indicate that the control values have not changed since being last persistently stored and then returns. In step 414, the method invokes the method SetActive of this CPropPagePersistent object. In step 415, the method sets the initialization flag to indicate that the property page has been initialized and then returns. In step 416, the method invokes the method QueryCancel of this
25 CPropPagePersistent object and then returns.

Figure 5 is a flow diagram of the method DoCommand of the CPropPagePersistent class. The method sets the change flag if a value of a control has changed. In step 501, if a control value has changed as indicated by

the message received by the property page window procedure, then the method sets the change flag in step 502. The method then returns.

Figure 6 is a flow diagram of the method SetActive of the CPropPagePersistent class. This method manages the retrieving of the persistently stored value for each control of the property page, the initializing of the controls of the property page with the retrieved values, the initializing of the list boxes, and the displaying of the controls. Initially, the method checks the initialize flag to determine whether this CPropPagePersistent object has already been initialized and returns if it has been. In steps 601-604, the method loops creating a CPersistentControl object for each control of the property page of the appropriate subclass. In step 601, the method selects a control of the property page starting with the first. In step 602, if all the controls have already been selected, then the method continues at step 605, else the method continues at step 603. In step 603, the method creates a CPersistentControl object of the appropriate subclass for the type of the selected control. In step 604, the procedure adds the created CPersistentControl object to the list of control objects maintained by the CPropPagePersistent object. In steps 605-607, the method loops selecting each control object and directing the CPersistentControl object to load its value from persistent storage. In step 605, the method selects the next CPersistentControl object starting with the first. In step 606, if all the CPersistentControl objects have already been selected, then the method continues at step 608, else the method continues at step 607. In step 607, the method invokes the method Load of the selected CPersistentControl object and loops to step 605 to select the next CPersistentControl object. In step 608, the method loads from the registry all the values that were combined together to store with a single key for the property page. In step 609, the method invokes the method Display of this CPropPagePersistent object to display the controls and then returns.

Figure 7 is a flow diagram of the method Display of the CPropPagePersistent class. The method loops selecting each CPersistentControl object and invokes its method Display. In step 701, the method selects the next CPersistentControl object starting with the first. In step 702, if all the CPersistentControl objects have already been selected, then the method returns, else the method continues at step 703. In step 703, the method invokes the method Display of the selected CPersistentControl object and loops to step 701 to select the next CPersistentControl object.

Figure 8 is a flow diagram of the method Apply of the CPropPagePersistent class. The method loops selecting each CPersistentControl object and invokes its method Save to persistently store its value. In step 801, the method selects the next CPersistentControl object starting with the first. In step 802, if all the CPersistentControl objects have already been selected, then the method continues at step 804, else the method continues at step 803. In step 803, the method invokes the method Save of the selected CPersistentControl object and loops to step 801 to select the next CPersistentControl object. In step 804, the method stores in the registry all the values combined together and then returns. When the method Save is invoked in step 803 for a combobox or a button control, that method Save stored the control value in a registry value structure 304 so that the CPropPagePersistent object could access and combine the control values for persistent storage.

Figure 9 is a flow diagram of the method Load of the CPersistentEditControl class. This method parses the text of the buddy control, if present, to determine the key for this control. The method then retrieves the value from the registry and initializes the control to that retrieved value. (The method Save of the CPersistentControl subclasses operate in an analogous manner to the method Load for the subclasses.) In step 901, the method retrieves the definition of the next control from the resource file. In step 902, if the next

control is a buddy control, then the method continues at step 903, else the method returns. In step 903, the method invokes the method GetAndParseBuddy of this CPersistentEditControl object to identify the key for this control. In step 904, the method retrieves the control value from the registry using the key. In step 905,
5 the method sets the text in the control to the retrieved value and returns.

Figure 10 is a flow diagram of the method GetAndParseBuddy of the CPersistentEditControl class. The method retrieves the text associated with the buddy control and parses the text to identify the key. In step 1001, the method retrieves the text for the buddy control. In step 1002, the method parses
10 the key from the text. In step 1003, the method stores the key and returns.

Figure 11 is a flow diagram of the display method of the CPersistentEditControl class. This method simply returns. The display is handled when the text is loaded.

Figure 12 is a flow diagram of the method Load of the
15 CPersistentComboControl class. If a combobox has a buddy control, then that buddy control contains a list of the items for the list box of the combobox. In step 1201, the method retrieves the next control. In step 1202, if the next control is a buddy control, then the method continues at step 1203, else the method returns. In step 1203, the method retrieves the text for the buddy control. In step
20 1204, the method adds the identification of this CPersistentComboControl object to a registry value structure 304 so that the CPropPagePersistent object can retrieve its value from the combined values. The stored values for a combobox is preferably an index into the list of items.

Figure 13 is a flow diagram of the method Display of the
25 CPersistentComboControl class. This method fills the listbox based on the window text of the buddy control of this CPersistentComboControl object and sets its value to the value retrieved from the combined values by the CPropPagePersistent object. In step 1301, the method resets the edit box and list

box of the combobox. In step 1302, the method fills the list box with the list of items from the buddy control. In step 1303, the method retrieves the value for the edit box from a registry value structure 304. In step 1304, the method sets the retrieve value into the edit box and returns.

5 Figure 14 is a flow diagram of the method Load of the CPersistentButtonControl class. This method determines whether the control is a check box, and, if so, adds its identifier to a registry value structure 304 so that the CPropPagePersistent object can retrieve its value. In step 1401, the method retrieves the style of the window from the resource file. In step 1402, if the
10 control is a check box, then the method continues at step 1403, else the method returns. In step 1403, the method adds the identifier of the control to a registry value structure and returns.

 Figure 15 is a flow diagram of the method Display of the CPersistentButtonControl class. This method retrieves its value from a registry
15 value structure 304 and sets the check box accordingly. In step 1501, the method retrieves the registry value. In step 1502, the method sets the check box to the retrieved value and returns.

 Although the present invention has been described in terms of a preferred embodiment, it is not intended that the invention be limited to these
20 embodiments. Modifications within the spirit of the invention will be apparent to those skilled in the art. The scope of the present invention is defined by the claims that follow.

Claims

- 5 1. In a computer system, a method for specifying a user interface comprising at least one displayed control for display in a window, the method comprising:
at development time of the user interface, specifying a first control
definition in a format for defining a user interface element and defining the at least one
displayed control for display in a window, wherein the at least one displayed control has
10 a display method for displaying itself at runtime; and
at development time of the user interface, specifying a second control definition
in a format for defining a user interface element and defining information for use by the
display method of the displayed control specified by the first control definition at run
time of the user interface, wherein the second control definition defines a control not to
15 be presented in the user interface as a control at run time of the user interface.
2. A computer readable medium containing computer executable instructions for performing the method of claim 1.
- 20 3. The method of claim 1 wherein
the user interface is displayed by a computer program that controls
displaying the window; and
the second control definition defining information for assisting in presentation of
the displayed control specified by the first control definition at run time of the user
25 interface is modifiable to modify behavior of the at least one displayed control specified
by the first control definition for display in a window without modifying the computer
program that controls displaying the window.
- 30 4. The method of claim 1 wherein the second control definition comprises a text
parameter, wherein the text parameter comprises the information for assisting in
presentation of the displayed control specified by the first control definition at run time
of the user interface.
5. The method of claim 1 wherein the second control definition is specified by
35 defining an invisible, static, and disabled control.
6. The method of claim 1 wherein the information defined by the second control

definition for assisting in presentation of the displayed control specified by the first control definition at run time of the user interface comprises a list of choices associated with the displayed control specified by the first control definition for presentation in the user interface at run time.

5

7. The method of claim 6 wherein the first control definition defines a combobox control having a set of items for presentation in the window at run time of the user interface and the second control definition specifies the set of items.

10 8. A computer-implemented method for presenting a property page as part of a computer program, the method comprising:

at development time of the computer program, specifying in a resource file a first control definition defining a combobox for presentation on the property page at runtime of the computer program;

15 at development time of the computer program, in the resource file, immediately following the first control definition, specifying a second control definition defining an invisible, static, disabled control comprising information specifying choices to be presented in association with the combobox on the property page, wherein the invisible, static, disabled control is never presented as a control; and

20 at runtime of the computer program, presenting the property page with the combobox having the choices specified by the second control definition without ever presenting the invisible, static, disabled control as a control.

9. A computer readable medium containing computer executable instructions for
25 performing the method of claim 8.

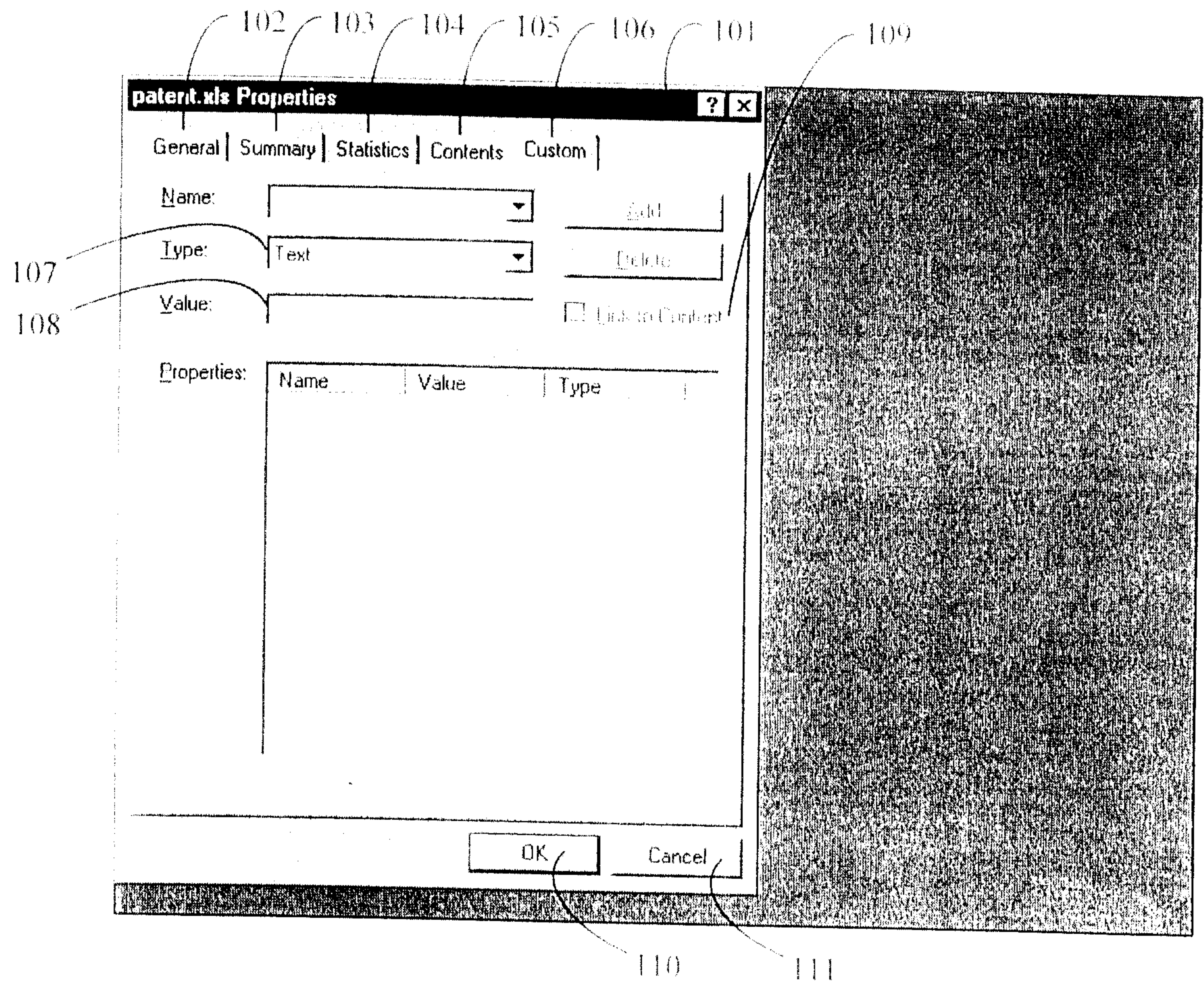


Fig. 1a
Prior Art

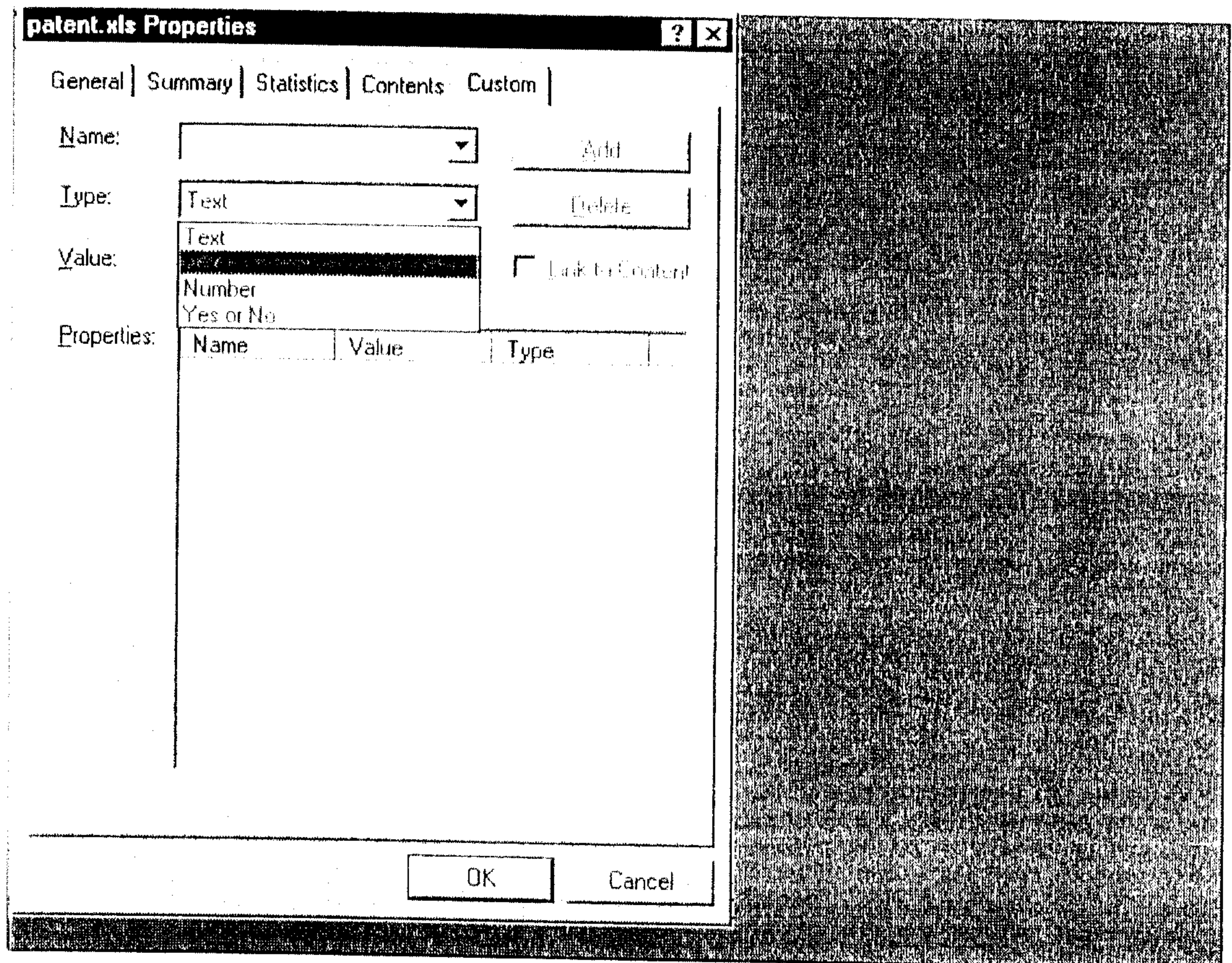


Fig. 1b
Prior Art

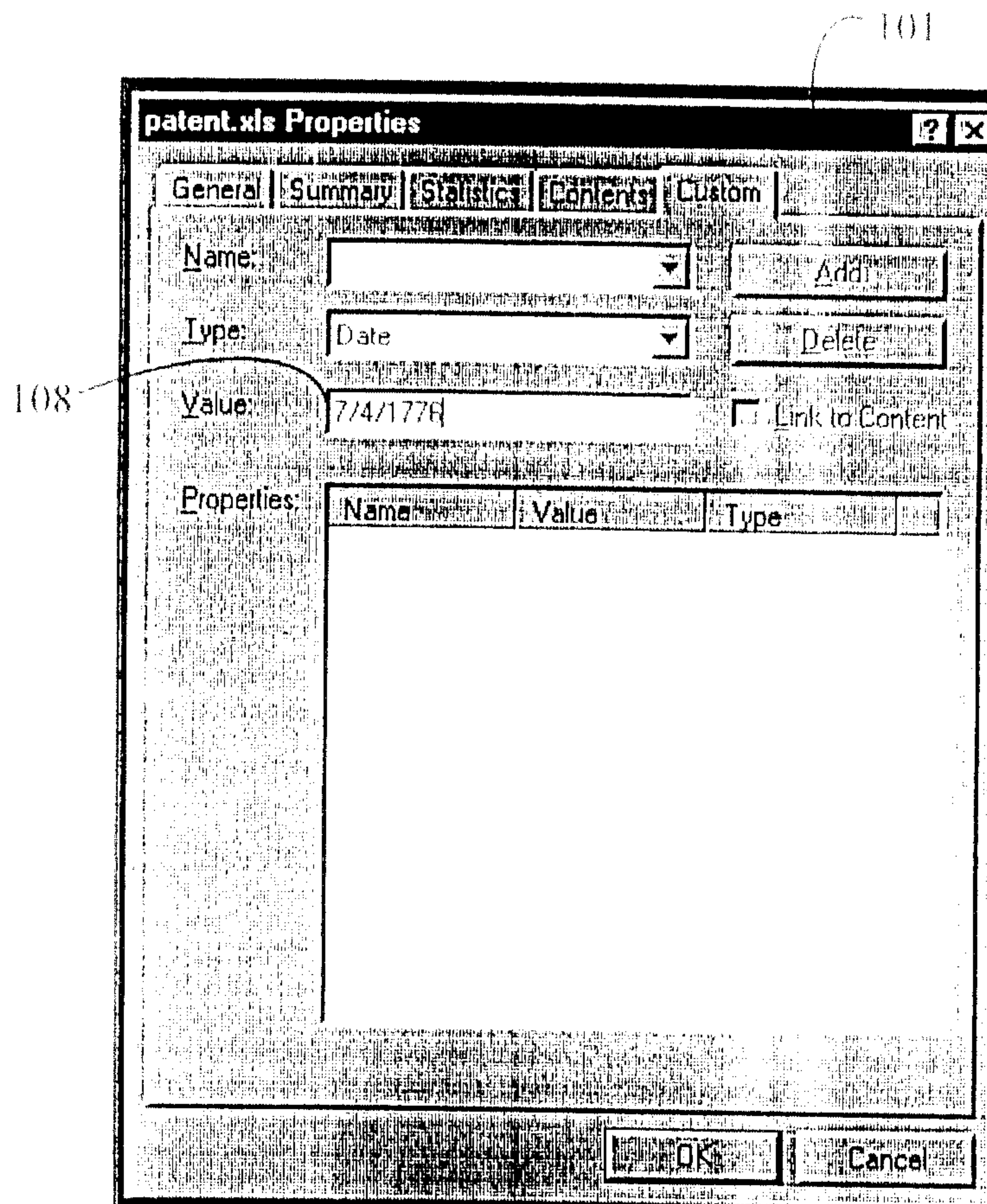


Fig. 1c
Prior Art

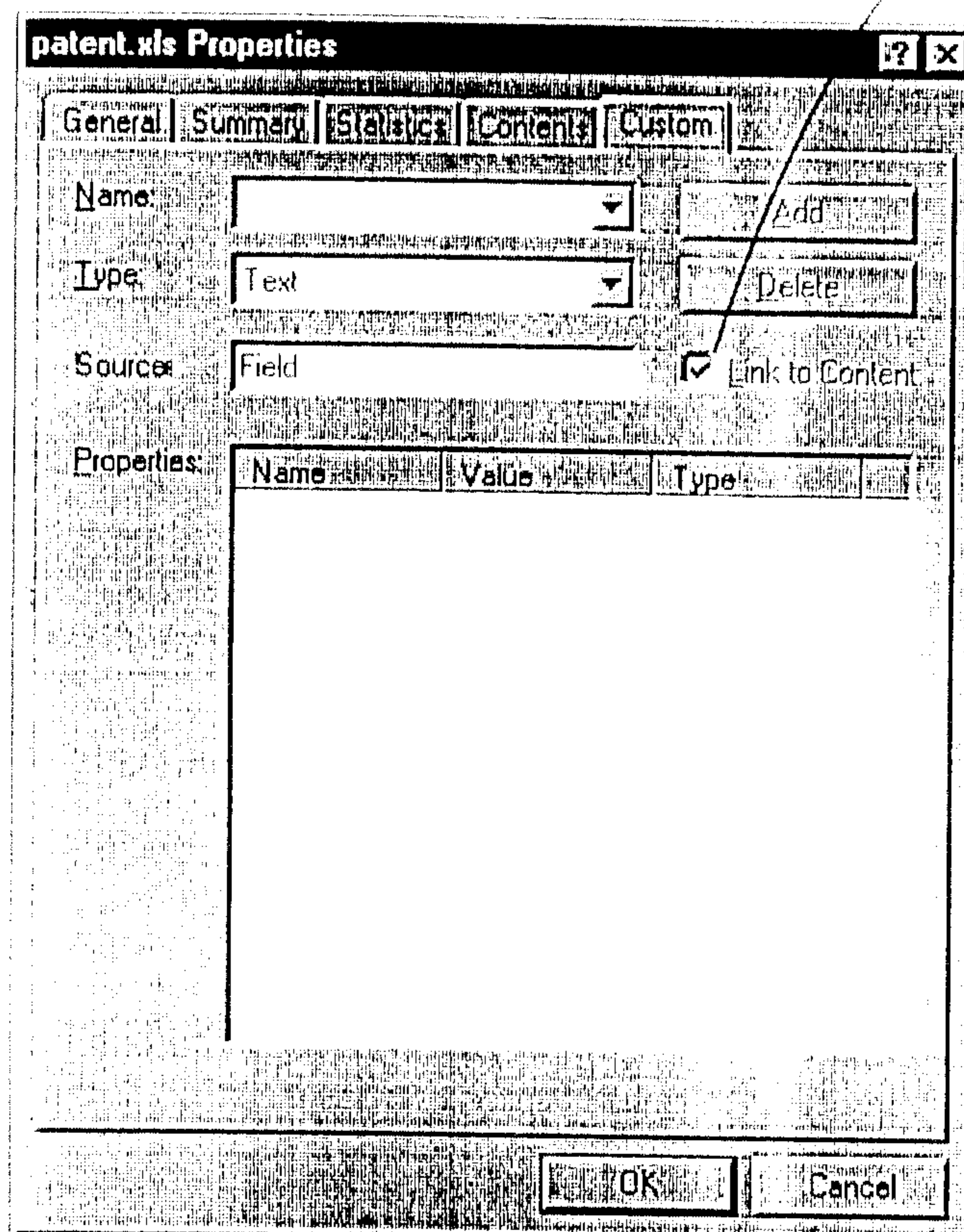


Fig. 1d
Prior Art

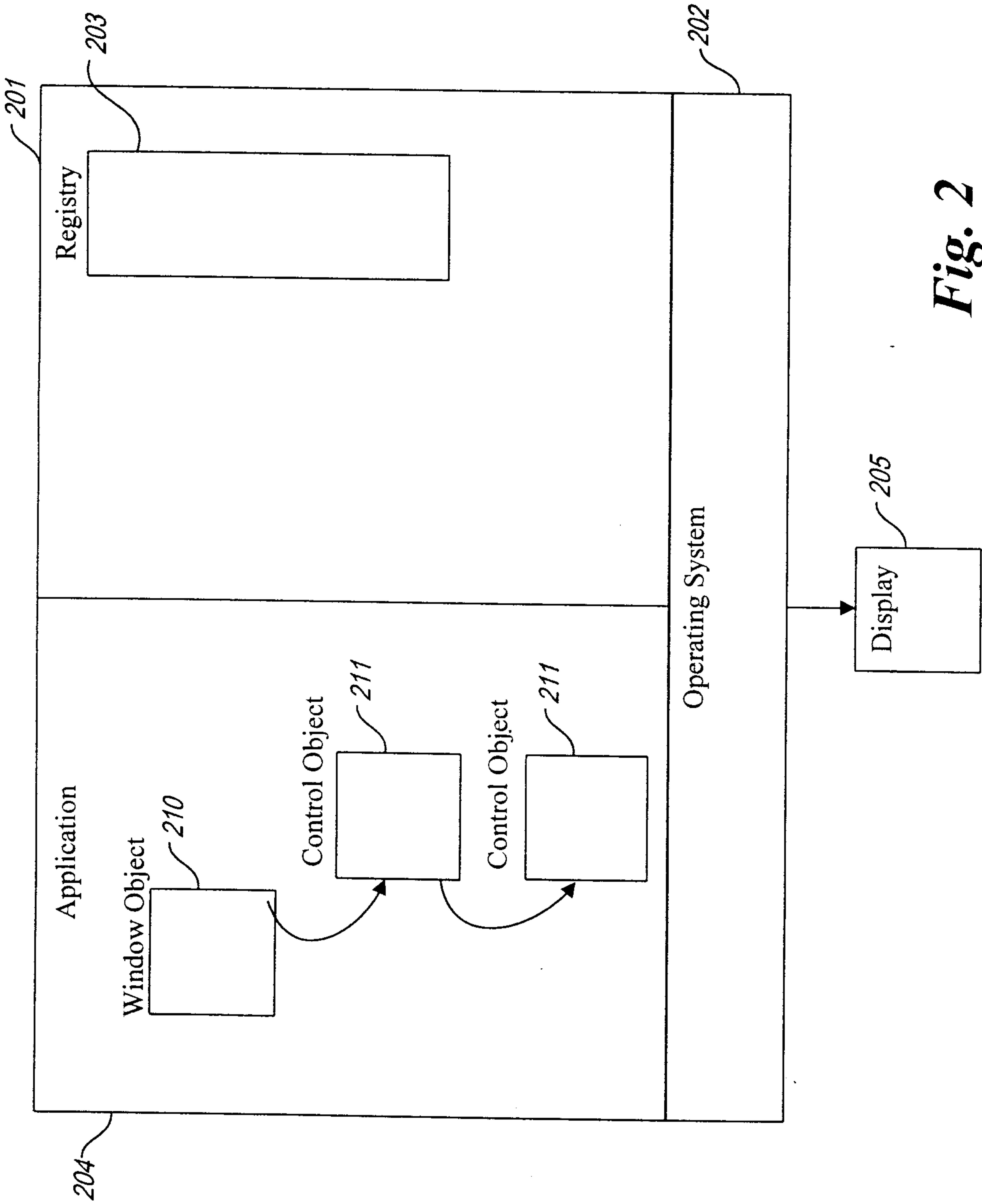


Fig. 2

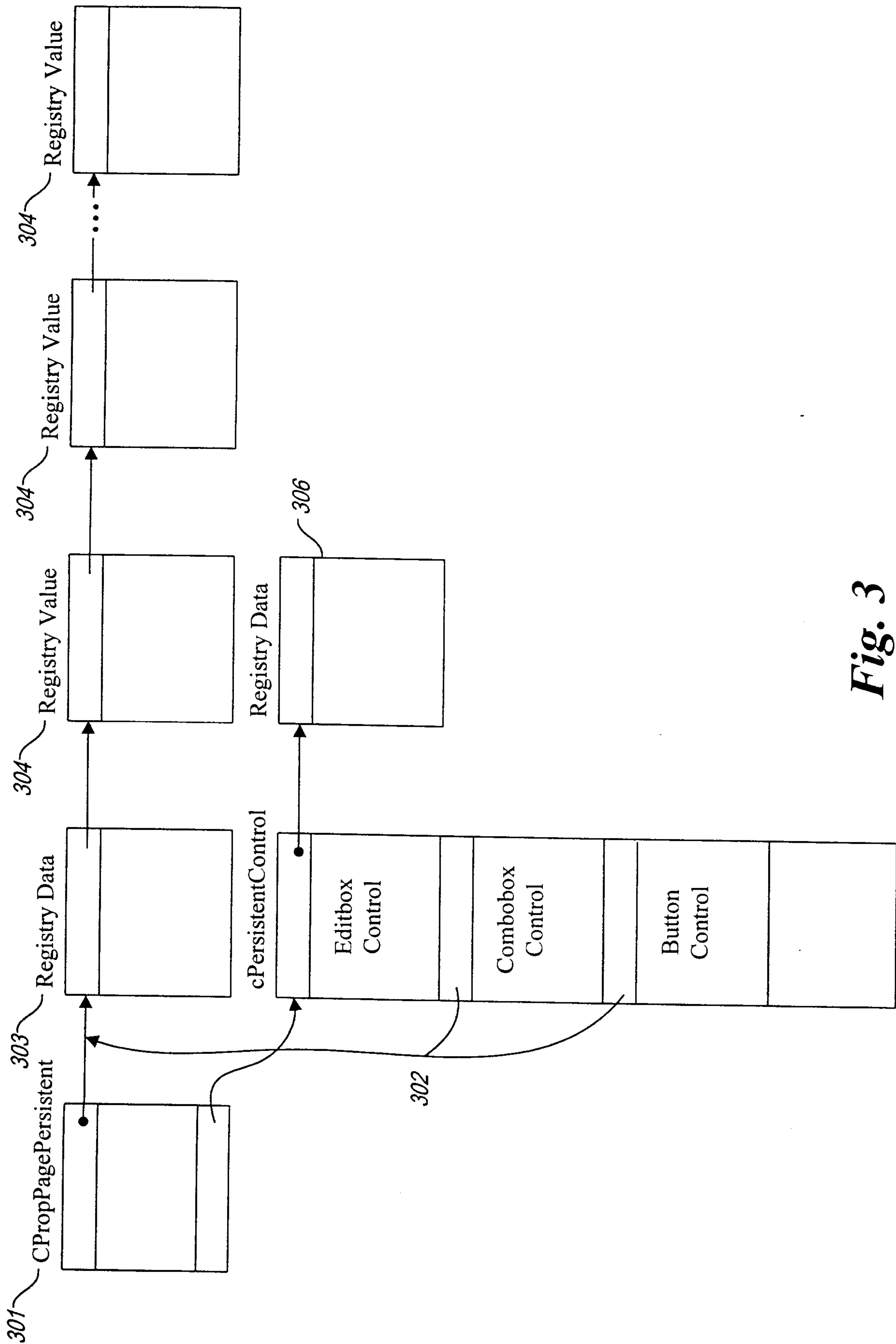
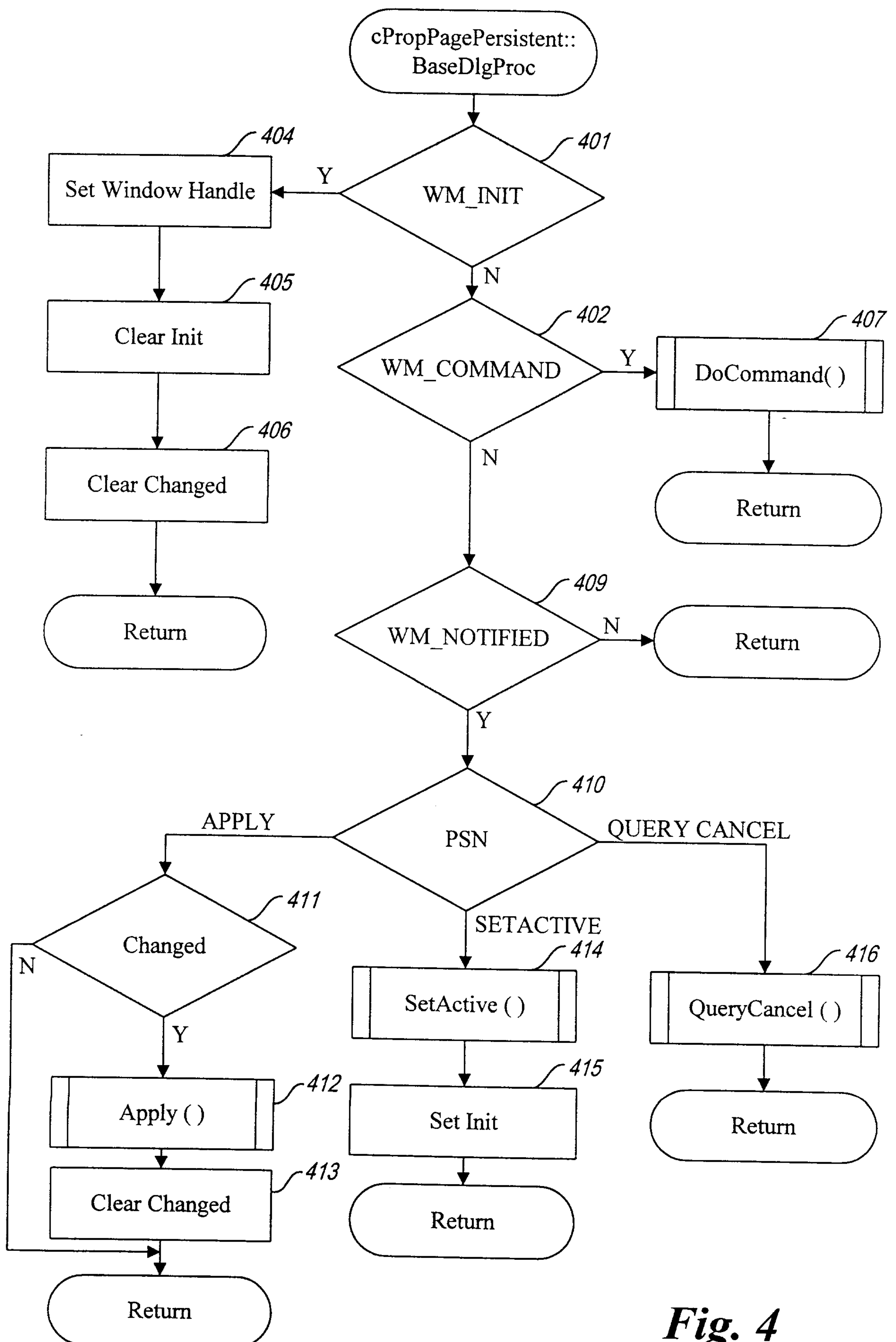
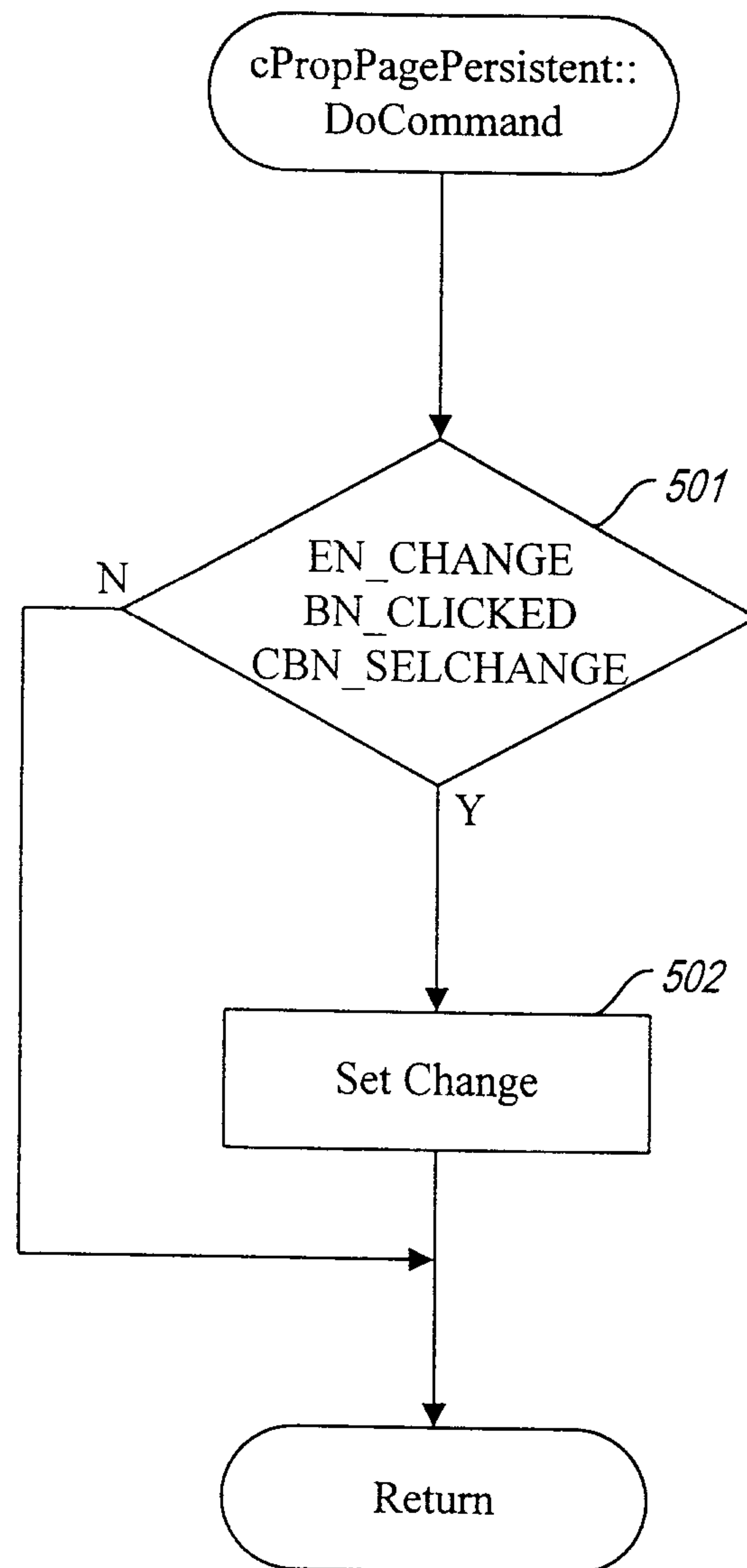
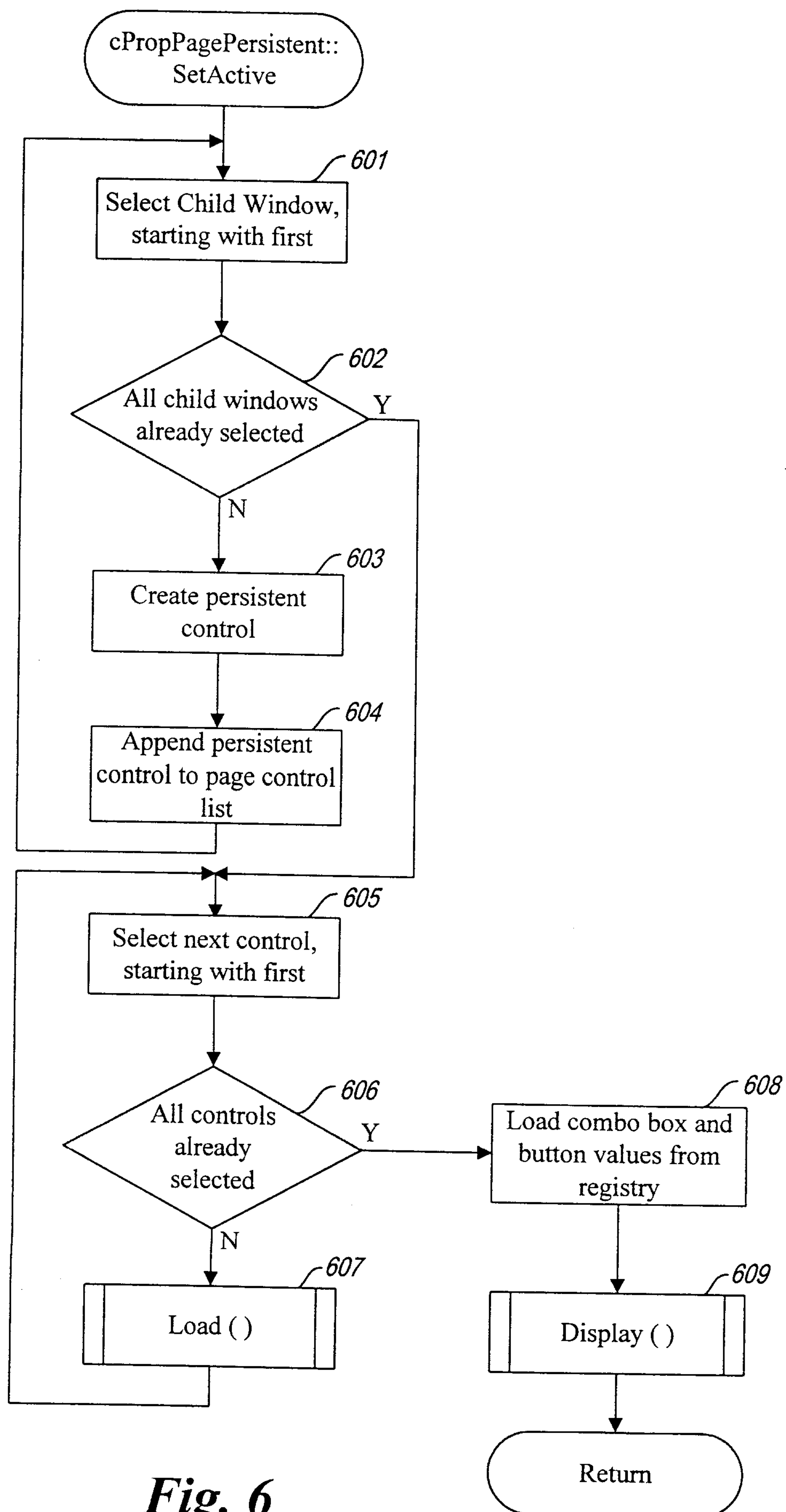
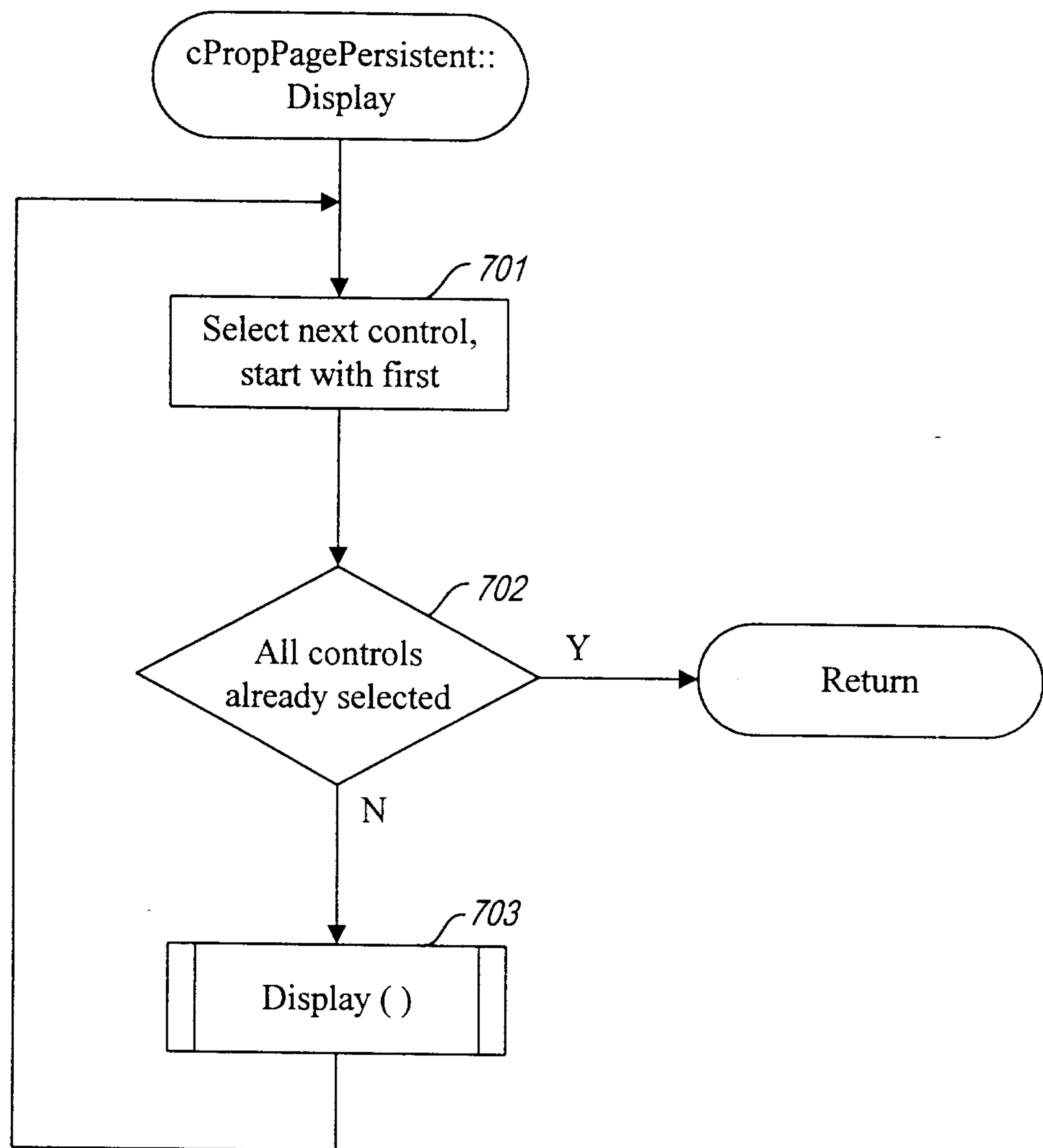


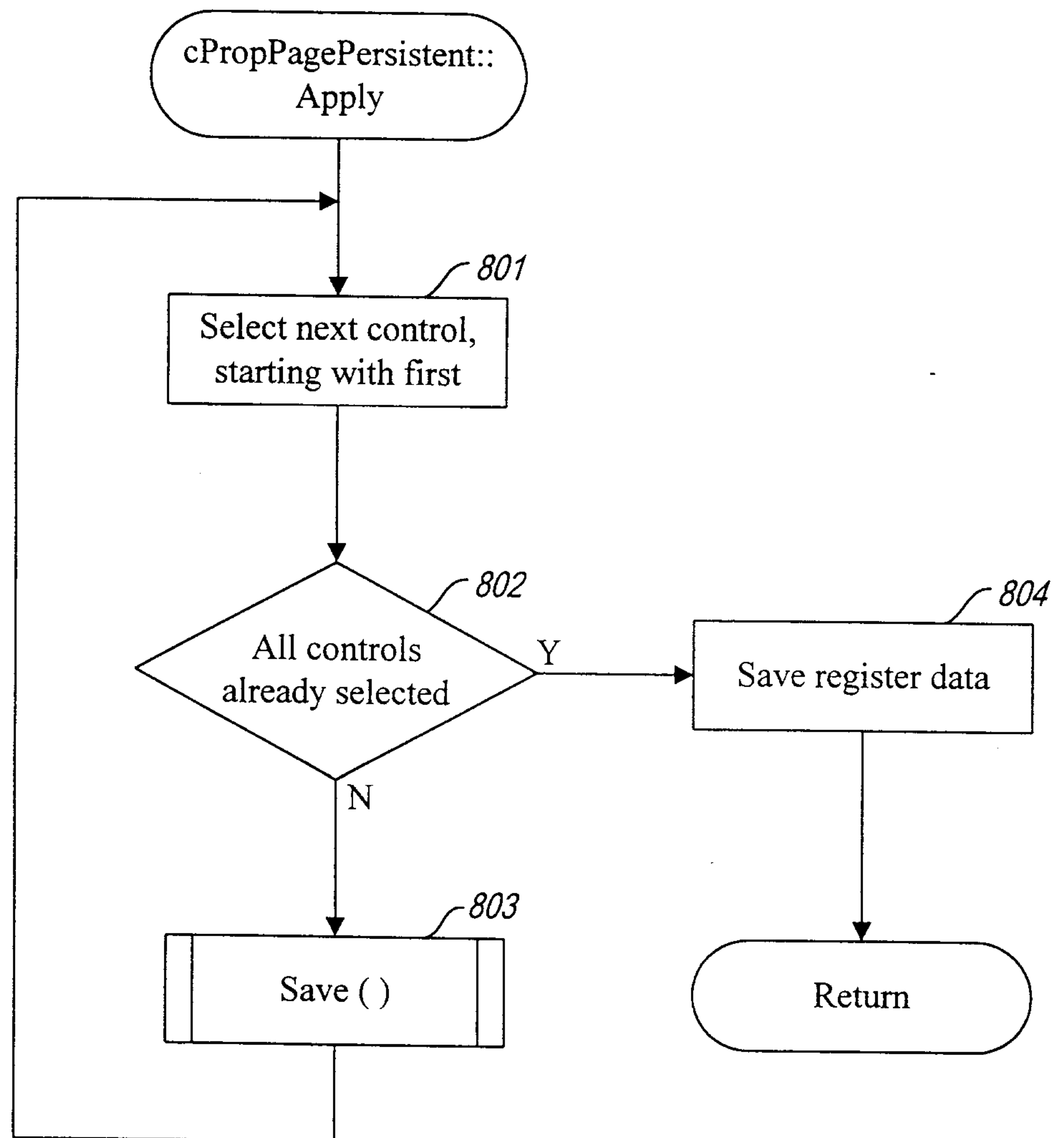
Fig. 3

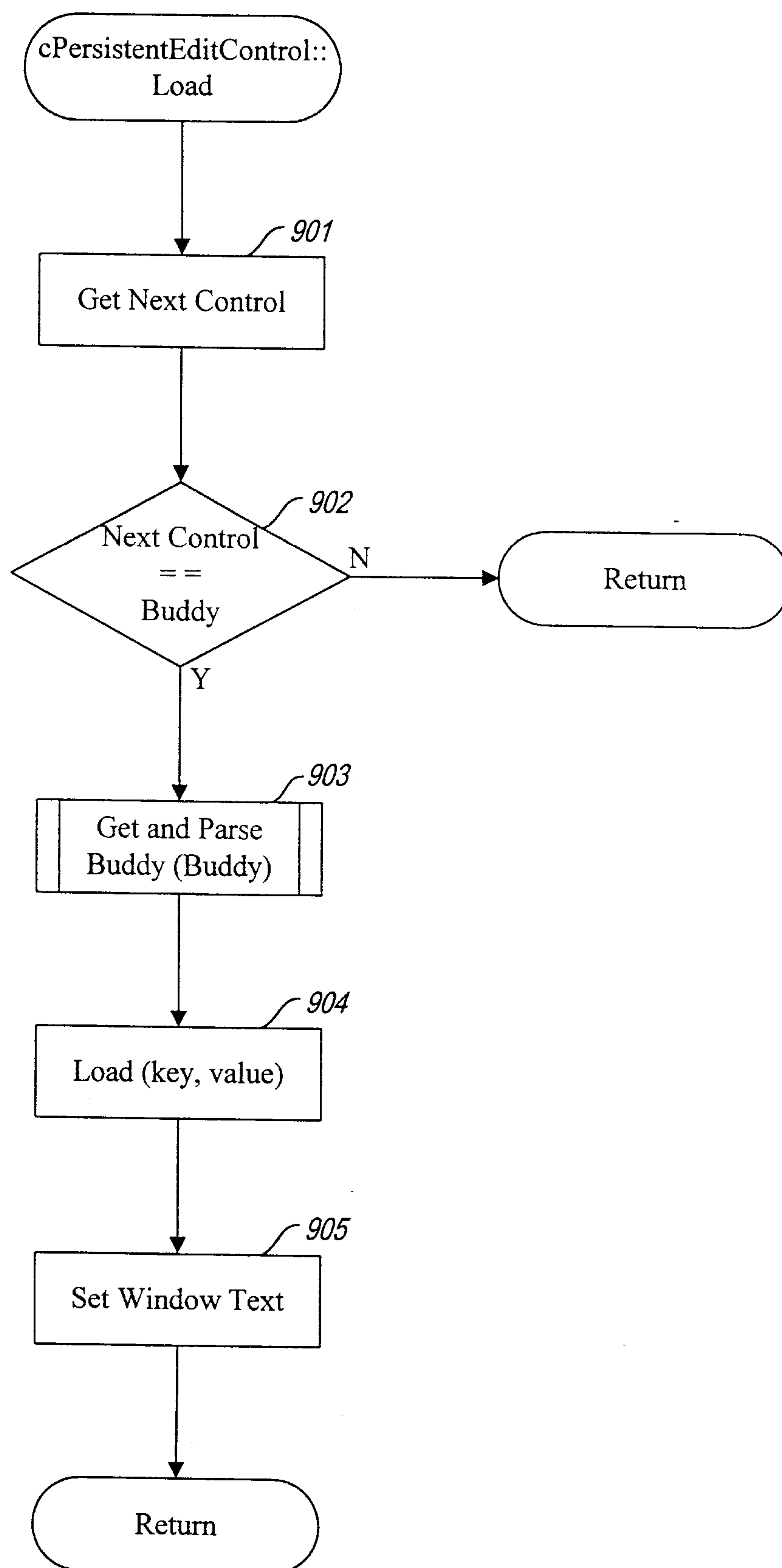
**Fig. 4**

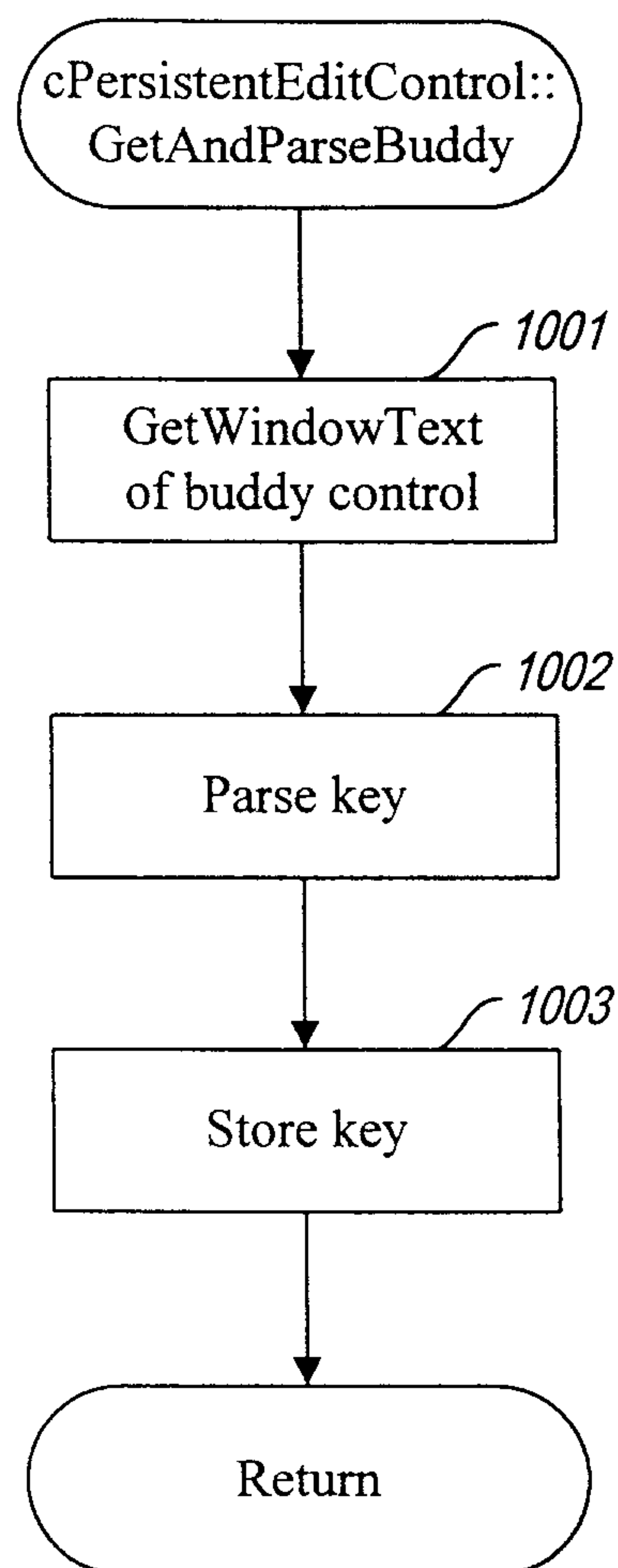
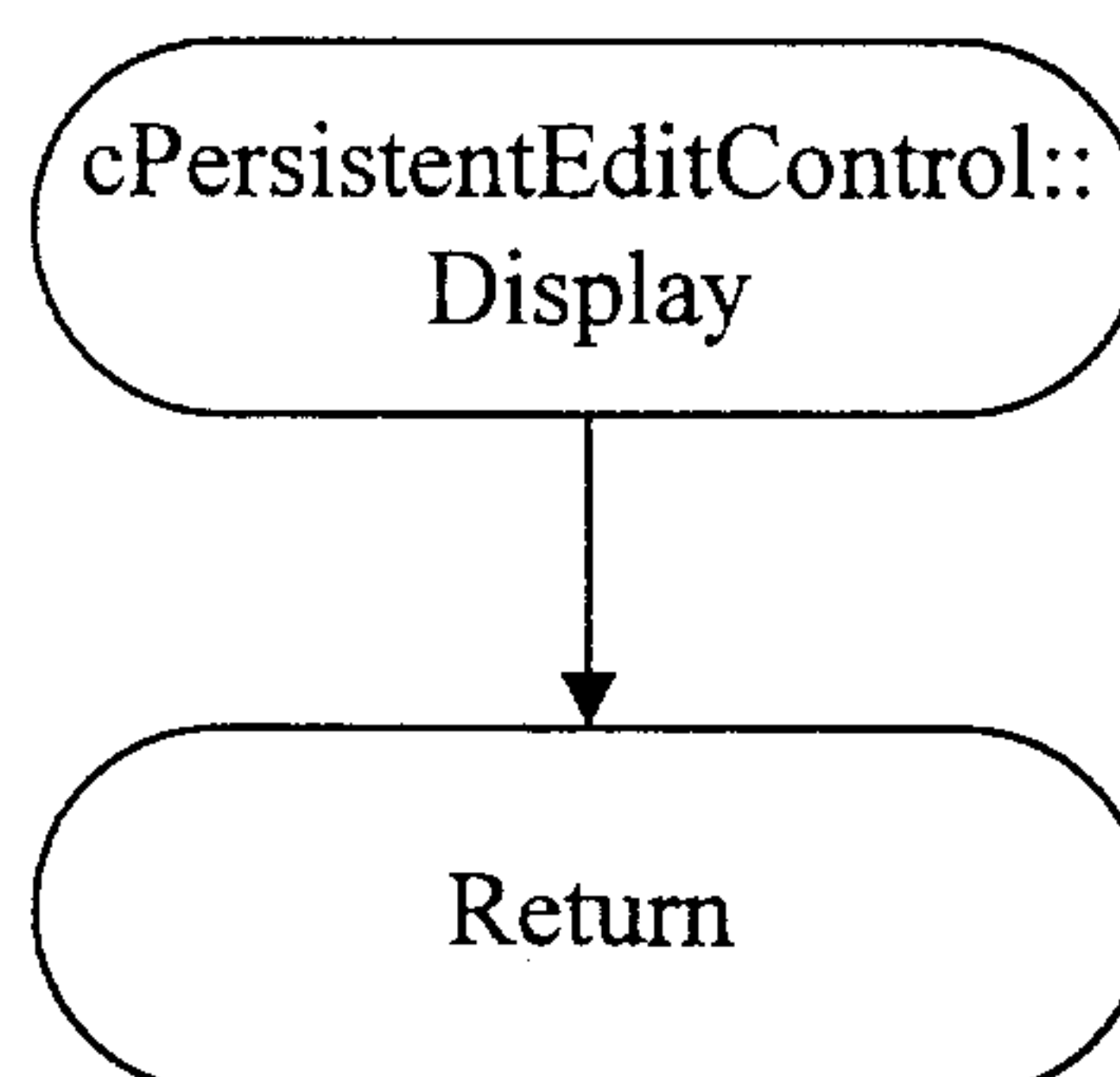
***Fig. 5***

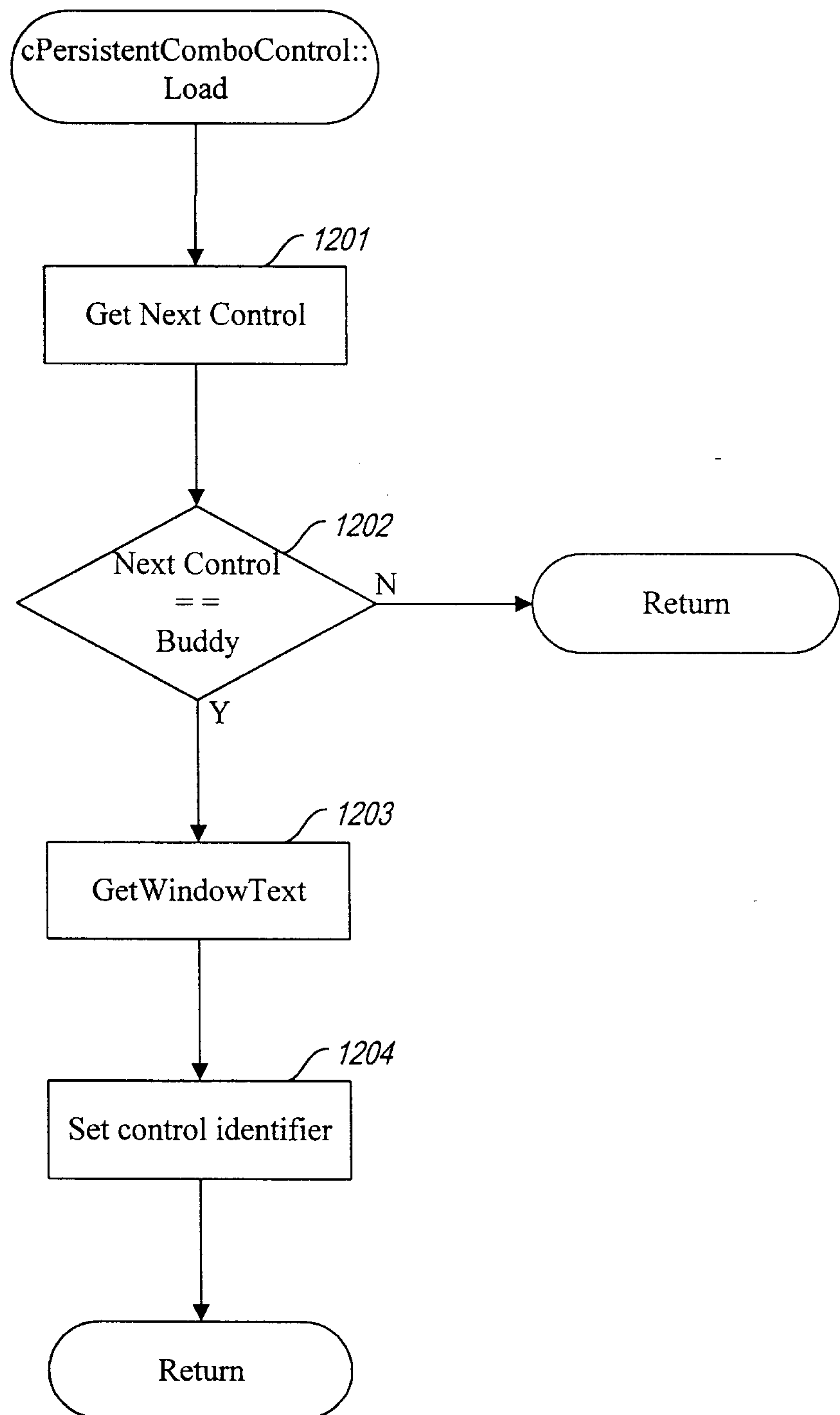
**Fig. 6**

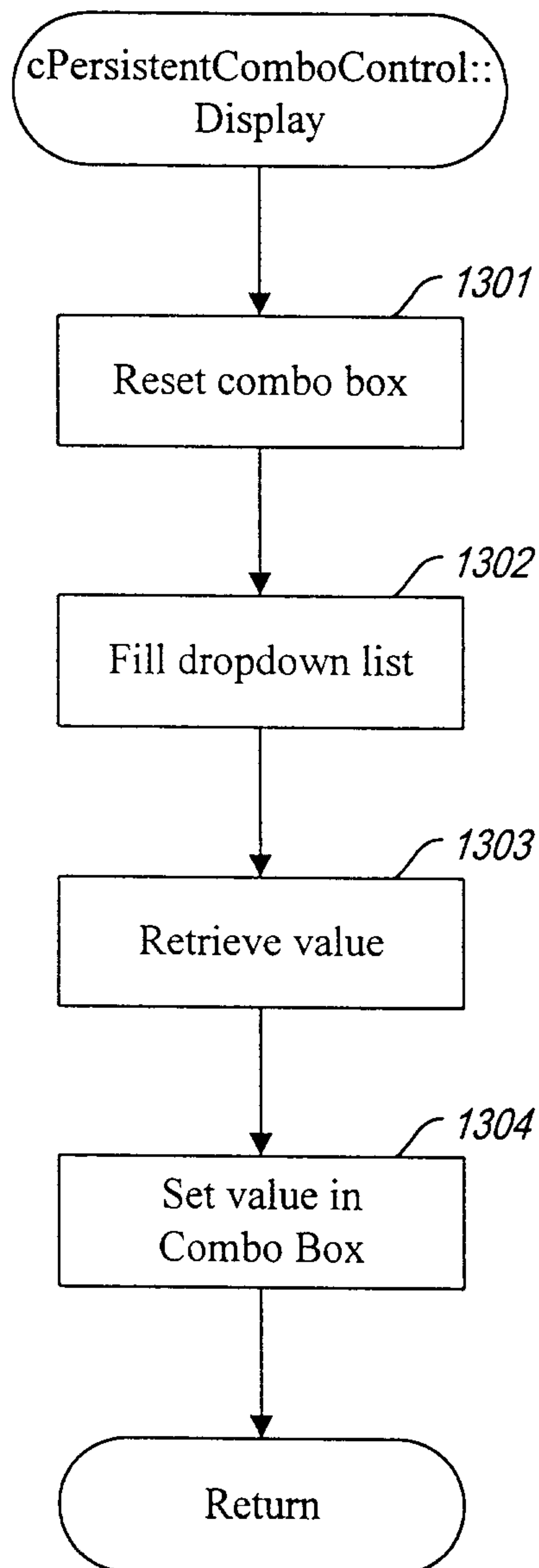
**Fig. 7**

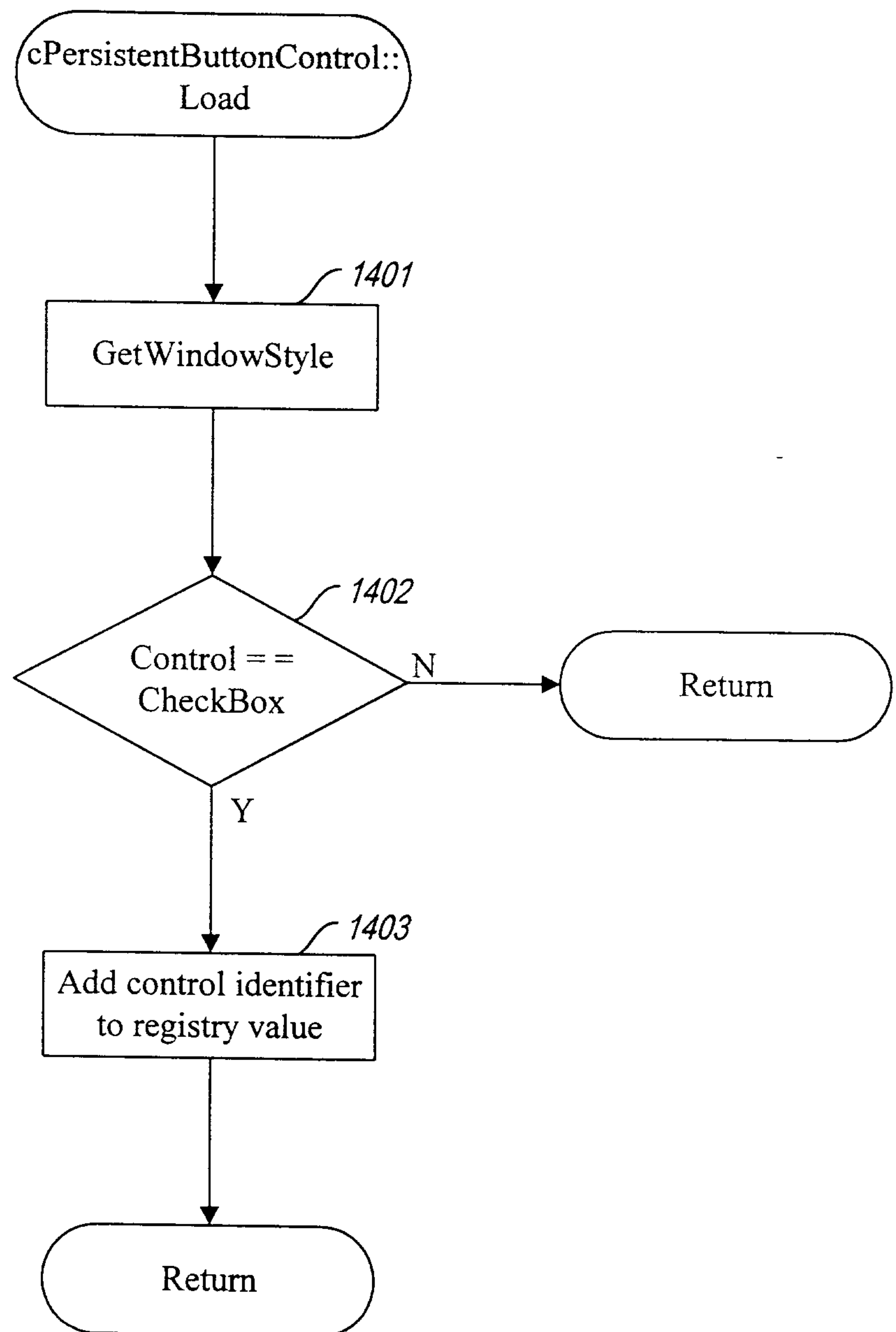
**Fig. 8**

**Fig. 9**

***Fig. 10******Fig. 11***

**Fig. 12**

***Fig. 13***

**Fig. 14**

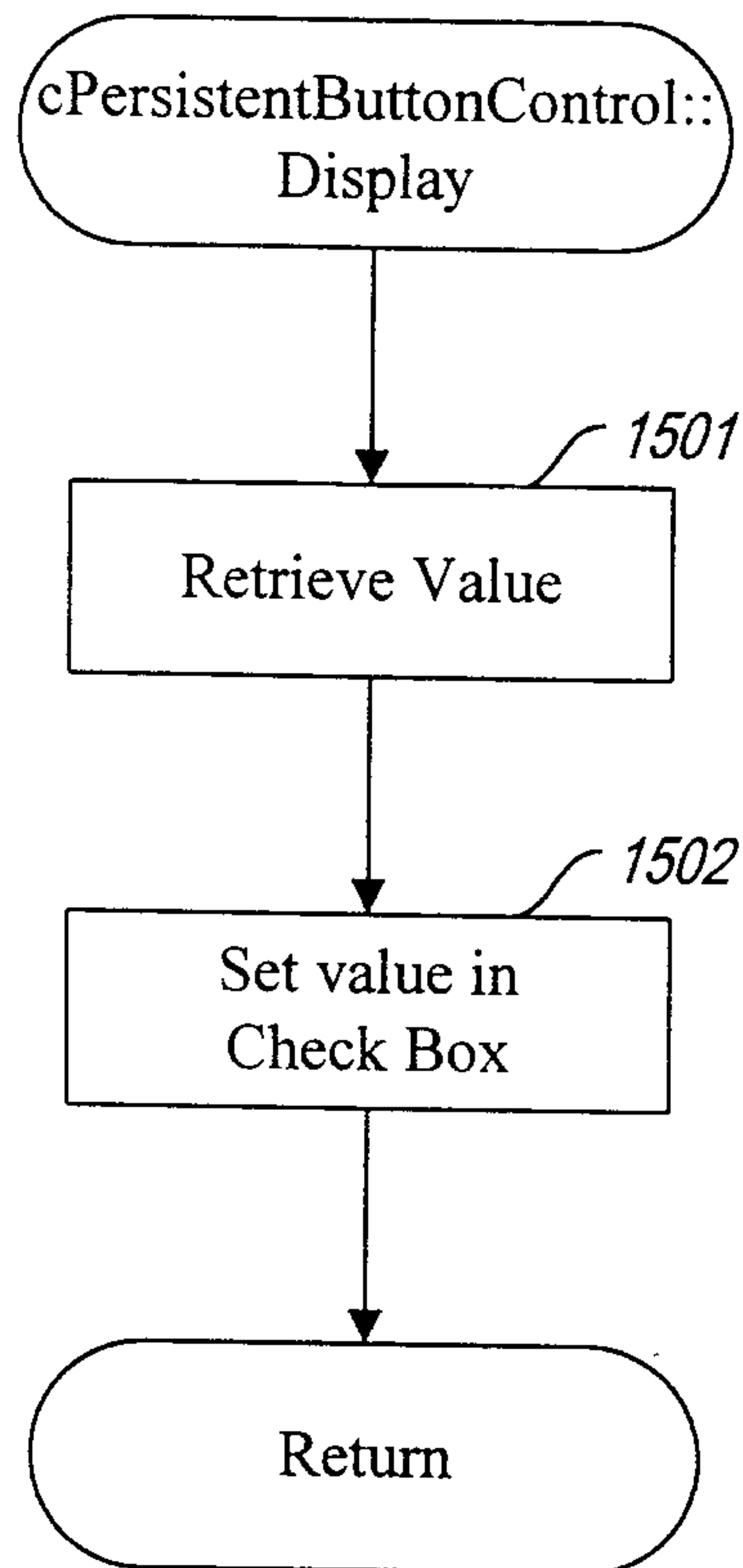


Fig. 15

***Oyen Wiggs
Green & Mutala***
PATENT AGENTS

