[redacted]

Class	Subclass	ISSUE CLASSIFICATION
707	513	

6061695

UTILITY SERIAL NUMBER	PATENT DATE	PATENT NUMBER
08/761699	MAY 0 9 2000	6061695

SERIAL NUMBER	FILING DATE	CLASS	SUBCLASS	GROUP ART UNIT	EXAMINER
[illegible]	[illegible]	707 ~~395~~	513 ~~500 274~~	2769	YOUNG

BENJAMIN W. SLIVKA, CLYDE HILL, WA; [illegible], [illegible], WA;
CHRISTOPHER RALPH BROWN, SEATTLE, WA; GEORGE [illegible], REDMOND, WA; SATOSHI
NAKAJIMA, REDMOND, WA; SANJAY [illegible], REDMOND, WA; [illegible],
SEATTLE, WA.

CONTINUING DATA********************
VERIFIED NONE JLY

FOREIGN/PCT APPLICATIONS**********
VERIFIED NONE JLY

Foreign priority claimed 35 USC 119 conditions met Verified and Acknowledged	□ yes ☒ no □ yes ☒ no JLY Examiner's Initials	AS FILED	STATE OR COUNTRY	SHEETS DRWGS.	TOTAL CLAIMS	INDEP. CLAIMS	FILING FEE RECEIVED	ATTORNEY'S DOCKET NO.
			WA	7	17	2	[illegible]	[illegible]

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON
ONE WORLD TRADE CENTER SUITE 1600
121 SW SALMON STREET
PORTLAND OR 97204-2988

~~OPERATING SYSTEM SHELL [illegible] Document [illegible] user~~
~~[illegible]~~ OPERATING SYSTEM SHELL ~~[illegible]~~ HAVING A WINDOWING GRAPHICAL USER INTERFACE
WITH A DESKTOP DISPLAYED AS A HYPERTEXT MULTIMEDIA DOCUMENT

U.S. DEPT. OF COMM./PAT. & TM—PTO-436L (Rev.12-94)

PARTS OF APPLICATION FILED SEPARATELY	M. Ehsor Applications Examiner

NOTICE OF ALLOWANCE MAILED		CLAIMS ALLOWED	
10-27-99	(JOHN LEONARD YOUNG) Assistant Examiner	Total Claims 17	Print Claim 1

ISSUE FEE			DRAWING		
Amount Due 1210.00	Date Paid 12/21/99	James R. Trammell Supervisory Patent Examiner Technology Center 2700 Primary Examiner	Sheets Drwg. 7	Figs. Drwg. 7	Print Fig. 2
Label Area		PREPARED FOR ISSUE	ISSUE BATCH NUMBER D-57		

WARNING: The information disclosed herein may be restricted. Unauthorized disclosure may be prohibited by the United States Code Title 35, Sections 122, 181 and 368. Possession outside the U.S. Patent & Trademark Office is restricted to authorized employees and contractors only.

Form PTO-436A
(Rev. 8/92)

(FACE)

08/761699

APPROVED FOR LICENSE □

INITIALS JAN 31 97 734

PATENT APPLICATION

08761699

CONTENTS

Date Entered or Counted			Date Received or Mailed
	1.	Application 7 papers.	
	2.	E. Dec	2-10-97
	3.	Dec & Fee	4-15-97
	4.	Prior Art	3/10/97
3-14-99	5.	Non-final rej (3 mos)	3-23-99
	6.	Ext of time (1)	7-27-99
	7.	Amdt A	7-27-99 cofm 7-23-99
10-26-99	8.	Ex. Amdt B	10-27-99
4-14-00	9.	Formal Drawings (7 shts) set 1	12-21-99
	10.		
	11.		
	12.		
	13.		
	14.		
	15.		
	16.		
	17.		
	18.		
	19.		
	20.		
	21.		
	22.		
	23.		
	24.		
	25.		
	26.		
	27.		
	28.		
	29.		
	30.		
	31.		
	32.		

(FRONT)

[redacted]

SEARCHED

Class	Sub.	Date	Exmr.
345	326	3-16-99	JY
395	200.57	3-16-99	JY
707	501	3-16-99	JY
UPDATE SEARCH:			
707	513	10-26-99	JLY
707	501	10-26-99	JLY
345	333	10-26-99	JLY
345	315	10-26-99	JLY
346	349	10-26-99	JLY

SEARCH NOTES

	Date	Exmr.
APS WORD SEARCH	3-15-99	JY
APS WORD SEARCH	3-16-99	JY
CLASS SEARCH SHOES	3-16-99	JY
LEXIS/NEXIS WORD SRCH	3-16-99	JY
UPDATE SEARCH:		
LEXIS/NEXIS WORDSRCHS	10-26-99	JLY
(SHOES) CLASS SEARCH	10-26-99	JLY

INTERFERENCE SEARCHED

Class	Sub.	Date	Exmr.
707	513	10-26-99	JLY
707	501	10-26-99	JLY
345	333	10-26-99	JLY
345	115	10-26-99	JLY

Staple Issue Slip Here

POSITION		ID NO.	DATE
CLASSIFIER		10	1-31-97
EXAMINER		352	2/5/97
TYPIST		357	05/21/97
VERIFIER			
CORPS CORR.			
SPEC. HAND		500	5-14-97
FILE MAINT.			
DRAFTING			

INDEX OF CLAIMS

Claim		Date							
Final	Original	3-17-99	10-26-99						
1	1	✓	=						
2	2	✓	=						
3	3	✓	=						
4	4	✓	=						
5	5	✓	=						
6	6	✓	=						
7	7	✓	=						
8	8	✓	=						
9	9	✓	=						
10	10	✓	=						
11	11	✓	=						
12	12	✓	=						
13	13	✓	=						
14	14	✓	=						
15	15	✓	=						
16	16	✓	=						
17	17	✓	=						
	18								
	19								
	20								
	21								
	22								
	23								
	24								
	25								
	26								
	27								
	28								
	29								
	30								
	31								
	32								
	33								
	34								
	35								
	36								
	37								
	38								
	39								
	40								
	41								
	42								
	43								
	44								
	45								
	46								
	47								
	48								
	49								
	50								

SYMBOLS

✓ .... Rejected
= .... Allowed
- (Through numberal) Canceled
+ .... Restricted
N .... Non-elected
I .... Interference
A .... Appeal
O .... Objected

Claim		Date							
Final	Original								
	51								
	52								
	53								
	54								
	55								
	56								
	57								
	58								
	59								
	60								
	61								
	62								
	63								
	64								
	65								
	66								
	67								
	68								
	69								
	70								
	71								
	72								
	73								
	74								
	75								
	76								
	77								
	78								
	79								
	80								
	81								
	82								
	83								
	84								
	85								
	86								
	87								
	88								
	89								
	90								
	91								
	92								
	93								
	94								
	95								
	96								
	97								
	98								
	99								
	100								

(LEFT INSIDE)

ERNMENT PRINTING OFFICE: 1998-440-769

NT NUMBER

ICATION SERIAL NUMBER
8/761,699

LICANT'S NAME (PLEASE PRINT)
SLIVKA et al.

REISSUE, ORIGINAL PATENT NUMBER

ORIGINAL CLASSIFICATION

CLASS	SUBCLASS
707	513

CROSS REFERENCE(S)

CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)			
345	333			
345	115			
707	501			

INTERNATIONAL CLASSIFICATION

G	0	6	F		17/21

GROUP ART UNIT
2764

ASSISTANT EXAMINER (PLEASE STAMP OR PRINT FULL NAME)
JOHN LEONARD YOUNG

PRIMARY EXAMINER (PLEASE STAMP OR PRINT FULL NAME)
Jim Trammell

TO 270
EV. 5-91)

ISSUE CLASSIFICATION SLIP

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE

US006061695A

United States Patent [19]

Slivka et al.

[11] **Patent Number: 6,061,695**

[45] **Date of Patent: May 9, 2000**

[54] **OPERATING SYSTEM SHELL HAVING A WINDOWING GRAPHICAL USER INTERFACE WITH A DESKTOP DISPLAYED AS A HYPERTEXT MULTIMEDIA DOCUMENT**

[75] Inventors: **Benjamin W. Slivka**, Clyde Hill; **Teresa Anne Martineau**, Kirkland; **Christopher Ralph Brown**, Seattle; **George Pitt**, Redmond; **Satoshi Nakajima**, Redmond; **Sankar Ramasubramanian**, Redmond; **Mike Sheldon**, Redmond, all of Wash.

[73] Assignee: **Microsoft Corporation**, Redmond, Wash.

[21] Appl. No.: **08/761,699**

[22] Filed: **Dec. 6, 1996**

[51] **Int. Cl.**^{7} **G06F 17/21**

[52] **U.S. Cl.** **707/513**; 345/333; 345/115; 707/501

[58] **Field of Search** 345/326, 333, 345/315, 349; 395/200.57; 707/501, 513

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,575,579	3/1986	Simon et al. .	
5,305,195	4/1994	Murphy .	
5,347,632	9/1994	Filepp et al. .	
5,491,820	2/1996	Belove et al. .	
5,572,643	11/1996	Judson .	
5,831,606	11/1998	Nakajima et al.	345/326
5,877,765	3/1999	Dickman et al.	345/349
5,905,492	5/1999	Straub et al.	345/333

FOREIGN PATENT DOCUMENTS

0749081A1	5/1996	European Pat. Off. .
WO 96/30864	10/1996	WIPO .

OTHER PUBLICATIONS

Gavron, Jacquelyn and Joseph Moran. "How to Use Microsoft (R) Windows (R) NT 4 Workstation." (Emeryville: Ziff Davis Press, 1996) pp. 24, 25, 28, 29, 88, 89, 100–105, 108, 109, 154–157 & 167, Jan. 1, 1996.

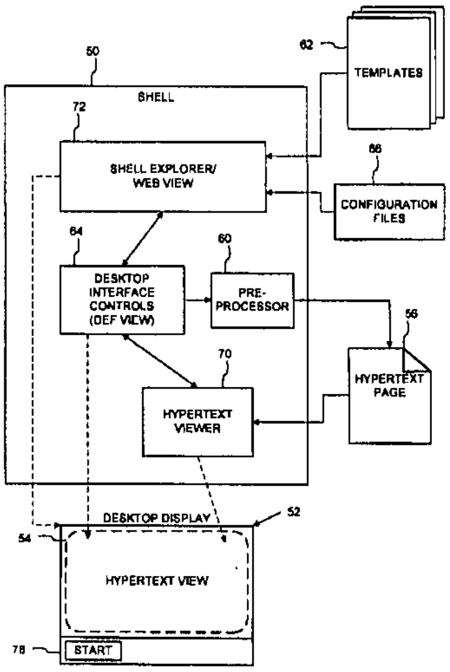
Tittle, Ed. and Steve James. "HTML for Dummies" 2d ed. (Foster City: IDG Books Worldwide, 1996) pp. 14, 15, 31–37, 57, 58, 60, 76–81, 131–133, 158, 350, 351, 385, 388, 392, 396 & 397, Mar. 11, 1996.

Primary Examiner—James P. Trammell
Assistant Examiner—John Leonard Young
Attorney, Agent, or Firm—Klarquist Sparkman Campbell Leigh & Whinston, LLP

[57] **ABSTRACT**

An operating system shell provides a graphical user interface having a windowing environment with a desktop. The shell synthesizes a hypertext page for display as the desktop in the graphical user interface. The hypertext page has an embedded software object which provides graphical icon-oriented and menu-driven user interface elements for activating operating system services in the displayed hypertext page. The shell also provides windowed hypertext pages for managing file system folders. The shell synthesizes the hypertext pages from templates which can be edited to incorporate a variety of multi-media enhancements with the user interface elements in the graphical user interface. Templates can be associated with specific folders in the file system to provide folder specific hypertext pages integrated with user interface elements for managing the folder.

17 Claims, 7 Drawing Sheets



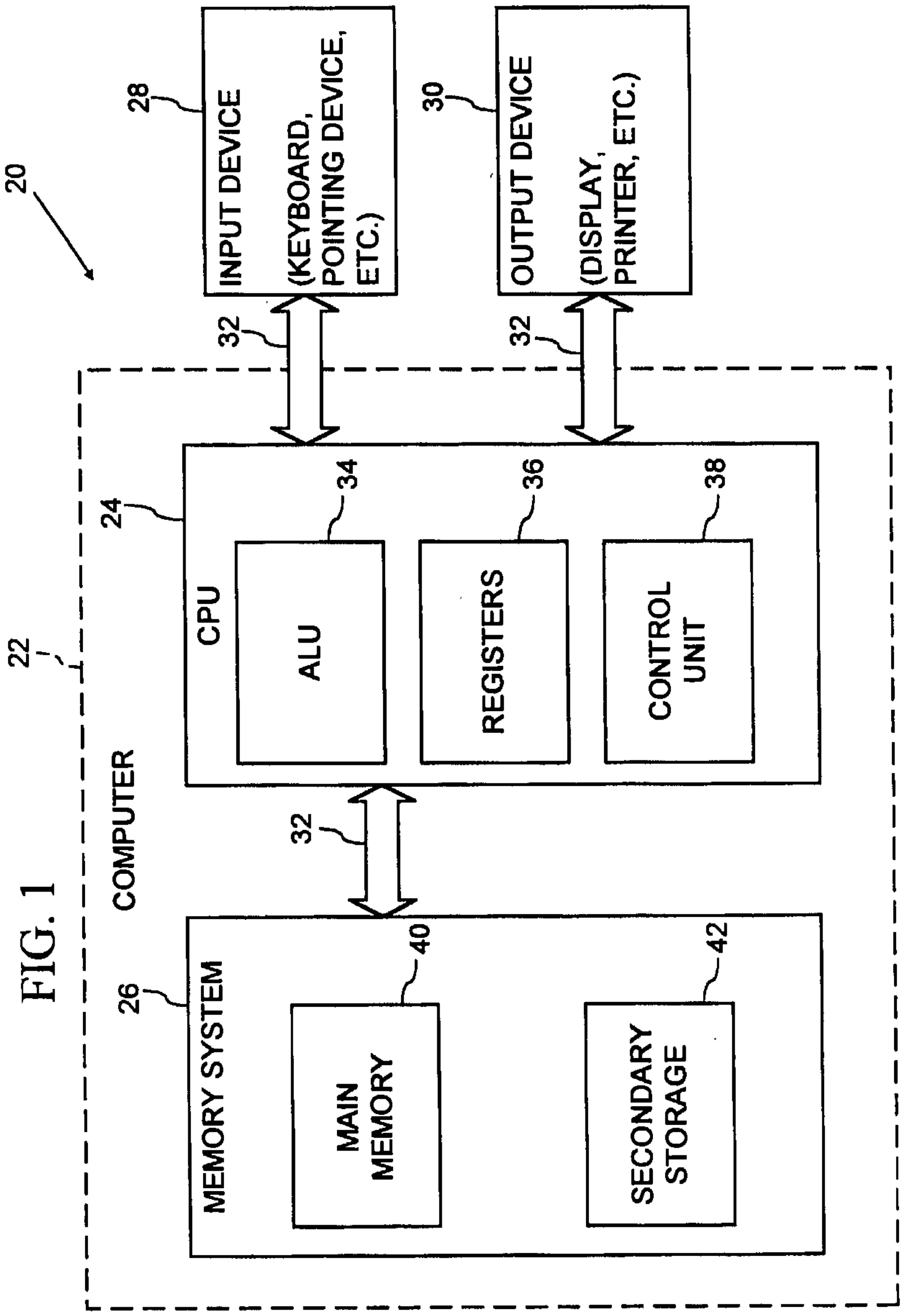
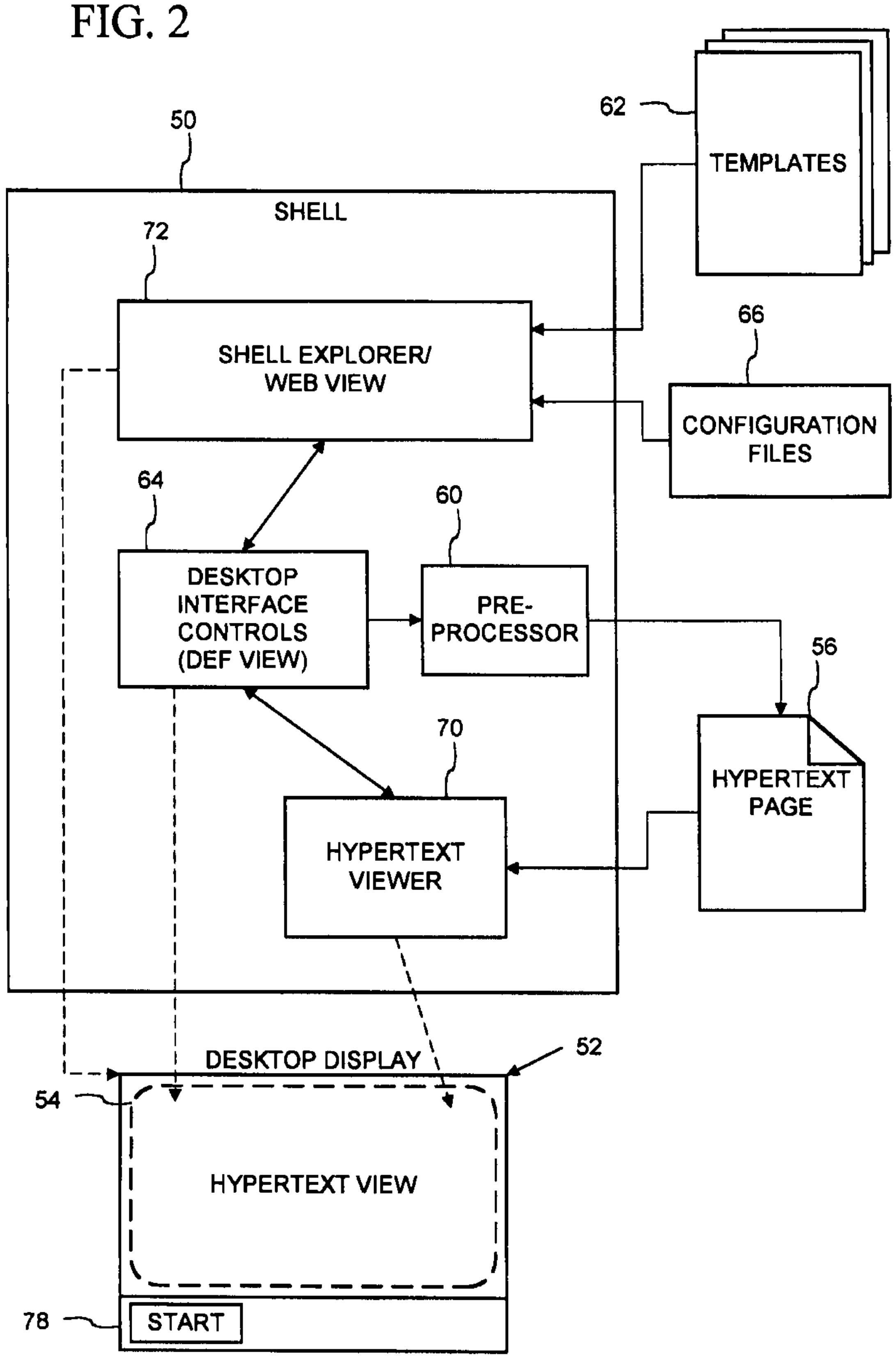

20
22
COMPUTER
24
CPU
34
ALU
36
REGISTERS
38
CONTROL UNIT
26
MEMORY SYSTEM
40
MAIN MEMORY
42
SECONDARY STORAGE
28
INPUT DEVICE
(KEYBOARD, POINTING DEVICE, ETC.)
30
OUTPUT DEVICE
(DISPLAY, PRINTER, ETC.)
32
32
32

FIG. 1

FIG. 2


50
SHELL
72
SHELL EXPLORER/ WEB VIEW
62
TEMPLATES
66
CONFIGURATION FILES
64
DESKTOP INTERFACE CONTROLS (DEF VIEW)
60
PRE-PROCESSOR
56
HYPERTEXT PAGE
70
HYPERTEXT VIEWER
DESKTOP DISPLAY
52
54
HYPERTEXT VIEW
78
START

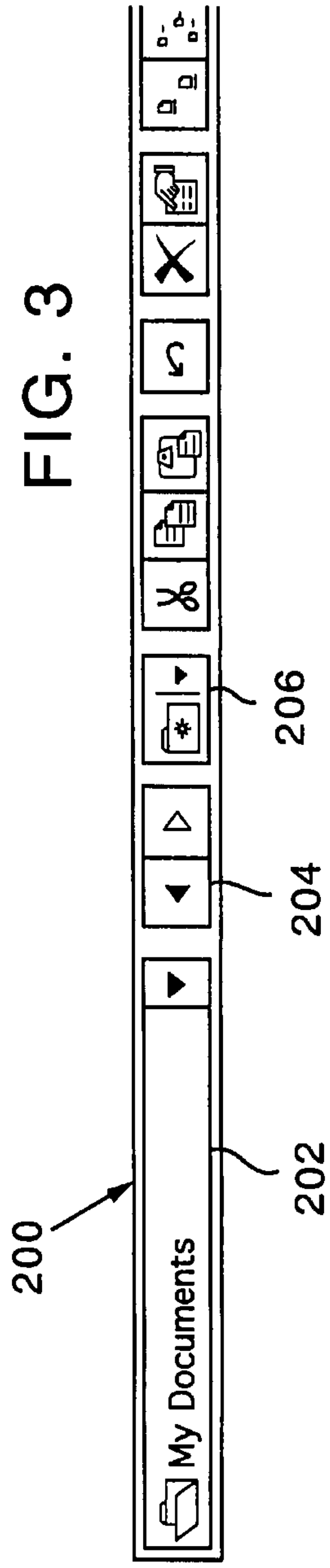


FIG. 3

FIG. 4

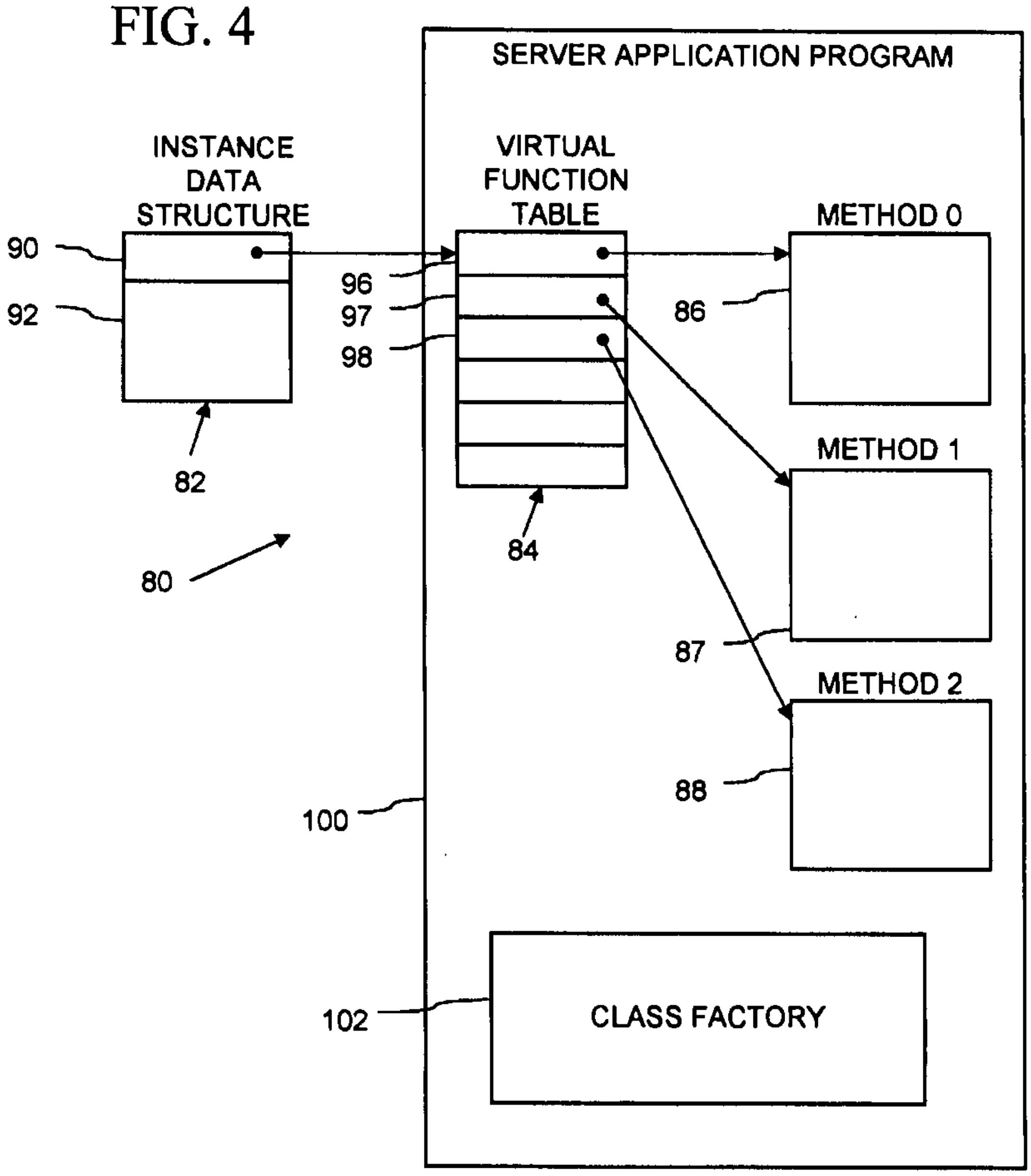

SERVER APPLICATION PROGRAM
INSTANCE DATA STRUCTURE
VIRTUAL FUNCTION TABLE
METHOD 0
METHOD 1
METHOD 2
CLASS FACTORY
90
92
82
80
96
97
98
84
86
87
88
100
102

FIG. 5

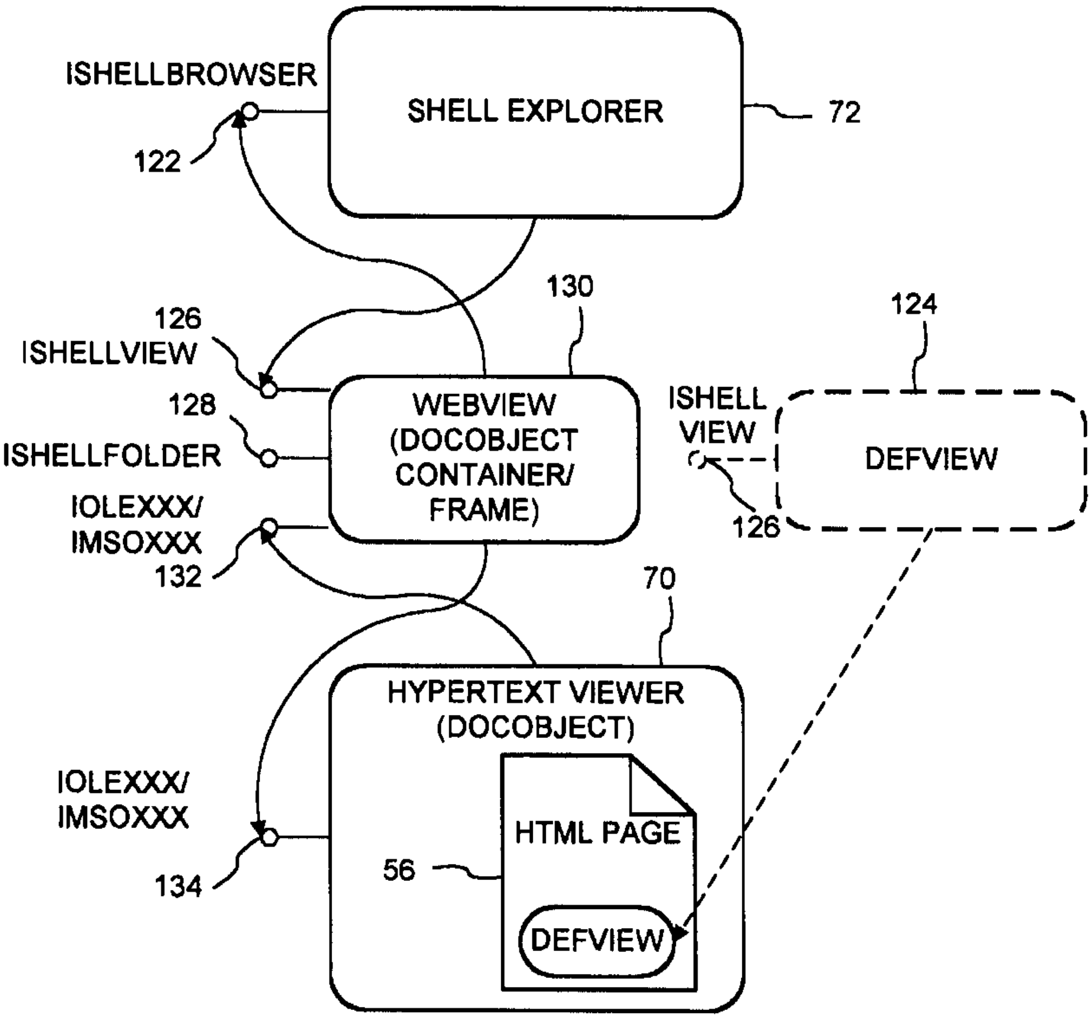
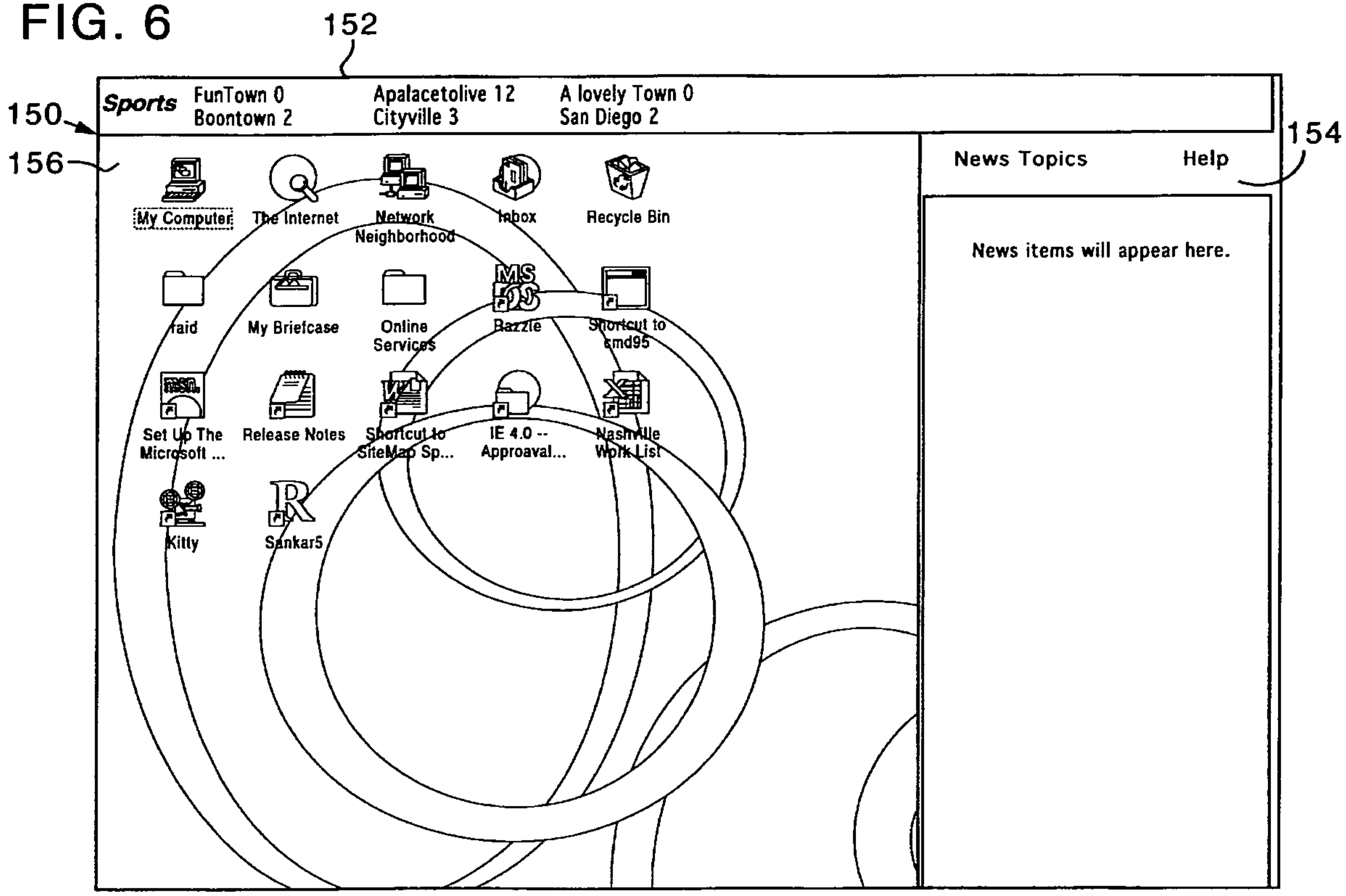

ISHELLBROWSER
SHELL EXPLORER
72
122
130
126
ISHELLVIEW
124
128
ISHELLFOLDER
WEBVIEW (DOCOBJECT CONTAINER/ FRAME)
ISHELL VIEW
DEFVIEW
126
IOLEXXX/ IMSOXXX
132
70
HYPERTEXT VIEWER (DOCOBJECT)
IOLEXXX/ IMSOXXX
HTML PAGE
56
134
DEFVIEW

FIG. 6

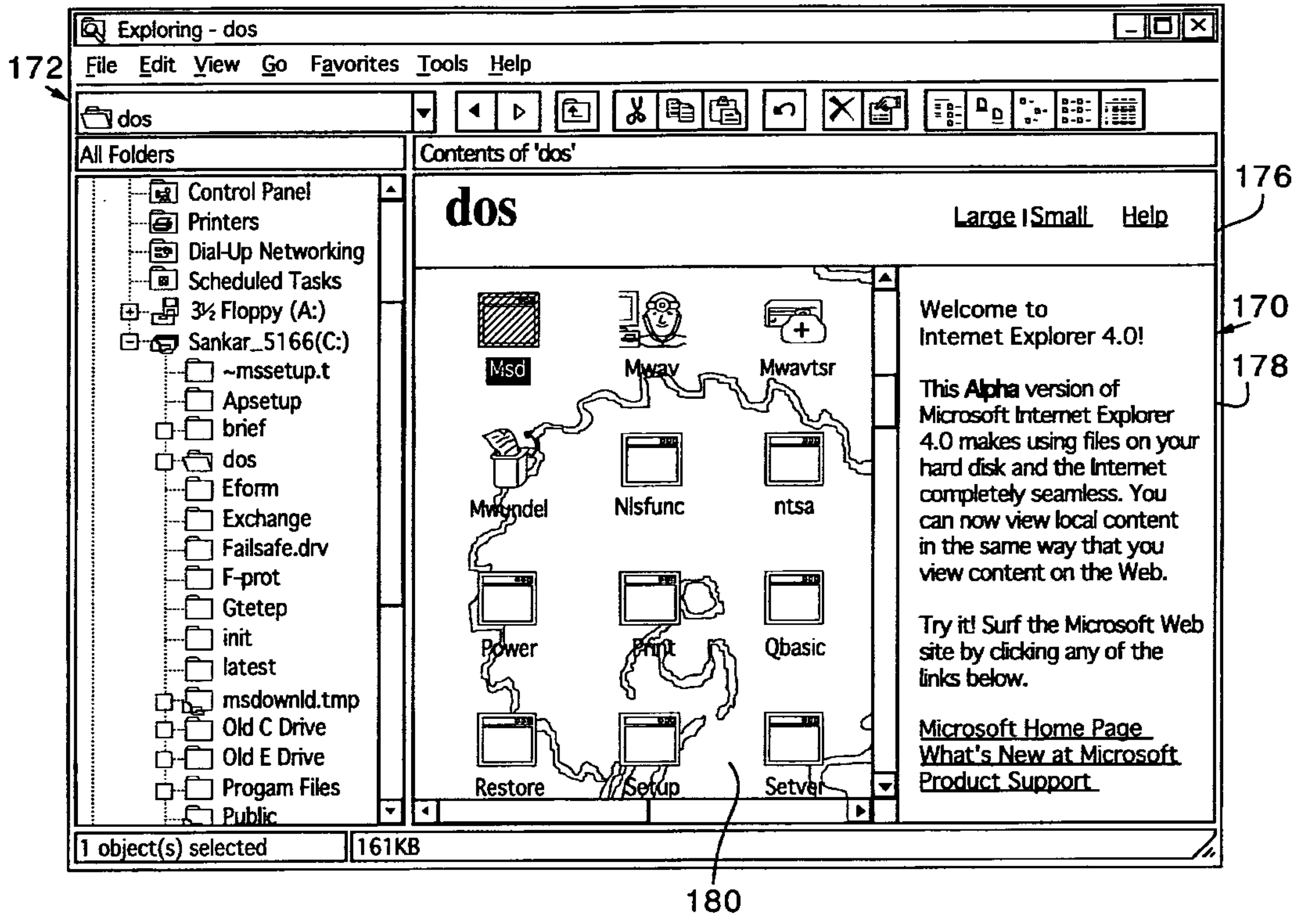


Sports FunTown 0 Boontown 2 Apalacetolive 12 Cityville 3 A lovely Town 0 San Diego 2

News Topics Help

News items will appear here.

FIG. 7



Welcome to
Internet Explorer 4.0!

This Alpha version of Microsoft Internet Explorer 4.0 makes using files on your hard disk and the Internet completely seamless. You can now view local content in the same way that you view content on the Web.

Try it! Surf the Microsoft Web site by clicking any of the links below.

Microsoft Home Page
What's New at Microsoft
Product Support

OPERATING SYSTEM SHELL HAVING A WINDOWING GRAPHICAL USER INTERFACE WITH A DESKTOP DISPLAYED AS A HYPERTEXT MULTIMEDIA DOCUMENT

FIELD OF THE INVENTION

This invention relates to a user interface or shell of an operating system, and more particularly relates to the incorporation of rich and dynamic multimedia content into such an interface.

BACKGROUND AND SUMMARY OF THE INVENTION

It is now common for operating systems to have a shell which provides a graphical user interface (GUI). The shell is a piece of software (either a separate program or component part of the operating system) that provides direct communication between the user and the operating system. The graphical user interface typically provides a graphical icon-oriented and/or menu driven environment for the user to interact with the operating system.

The graphical user interface of many operating system shells is based on a desktop metaphor. More specifically, the graphical user interface is intended to create a graphical environment which simulates working at a desk. These graphical user interfaces typically employ a windowing environment with a desktop.

The windowing environment presents the user with specially delineated areas of the screen called windows, each of which is dedicated to a particular application program, file or document. Each window can act independently, as if it were a virtual display device under control of its particular application program. Windows can typically be resized, moved around the display, and stacked so as to overlay another. In some windowing environments, windows can be minimized to an icon or increased to a full-screen display. Usually, the windows have a top to bottom order in which they are displayed, with top windows at a particular location on the screen overlaying any other window at that same location. The top-most window has the "focus" and accepts the user's input. The user can switch other windows to the top by clicking with a mouse or other pointer device, or by inputting certain key combinations. This allows the user to work with multiple application programs, files and documents in a similar manner as physically working with multiple paper documents and items which can be arbitrarily stacked or arranged on an actual desk.

The desktop of the graphical user interface is a screen display containing icons representing programs, files and resources available to the user. As such, the desktop acts as a launching point for running application programs, opening documents or files, and initiating operating system services. In accordance with the desktop metaphor, the desktop simulates the top of an actual desk on which various work items are made available to the office worker. The desktop in some graphical user interfaces thus includes icons representing resources found on a real desk, such as a file cabinet, telephone, wastebasket, and scratchpad, which are used to access their computer equivalents. In typical graphical user interfaces, the desktop always remains as a full-screen background display relative to the windowing environment. In other words, the desktop cannot be moved or resized, and all visible windows of the windowing environment overlay the desktop as would paper documents and other items on top of an actual desk.

A drawback to many prior desktops is their limited capacity to present multi-media content enhancements. For example, the shell of the Microsoft Windows® 95 operating system provides a graphical user interface with a windowing environment and a desktop. As a default, this desktop includes a "my computer" icon, a "network neighborhood" icon, and a "recycle bin" icon against a solid color background, and also includes a task bar along a bottom edge of the screen with a "start" button for menu-driven interaction. It is possible to add additional icons onto this desktop to represent other application programs, documents, files, and resources. The start menu also can be customized to include additional items, such as for launching particular applications, and opening documents or files. Further, a graphic image can be selected as a background of the desktop (called "wallpaper") in place of the default solid color background.

Other aspects of the Windows® 95 shell also are limited in their capacity to present multi-media content enhancements. In particular, the shell provides windowed folder views accessed through the my computer and network neighborhood icons that represent the contents (i.e., files and sub-folders) of a directory or folder in the file system of the computer or a connected local area network (LAN). The folder views have four display modes: large icon, small icon, list and detail. In the icon modes, the folder view displays icons representing the files and sub-folders in a window against a white background. The icons used to represent the files and sub-folders in the folder view are dependent on the type or format of the file, e.g., documents having a Microsoft Word format are represented with an icon indicative of that application program. In the list and detail modes, the folder view displays a text listing of the files and sub-folders in its window also against a white background.

An add-on product for the Microsoft Windows® 95 operating system, called the Microsoft Windows® 95 Plus Pack, includes packaged enhancements called "themes" for the graphical user interface of the Windows® 95 operating system. Each theme includes a group of resources which alter the appearance and feel of the graphical user interface. These resources include substitute icons, mouse pointer graphics and animations, sounds, a wallpaper, and a screen saver. A particular theme can be selected and applied to the graphical user interface using a themes applet which the plus pack installs into the Windows® 95 control panel (an application program group which includes small application programs or applets that control various aspects of the operating system). Again however, the themes provide only limited multi-media content enhancements to the desktop. The applied theme can change the desktop's wallpaper, and the graphics of the my computer, network neighborhood and recycle bin icons.

In contrast to the limited capacity for multi-media enhancement on the desktop, multi-media content commonly appears in a windowed application program in the graphical user interfaces of Windows® 95 and like operating system shells. Application programs which present multi-media content include desktop publishing, video games, multi-media encyclopedias and like references, Internet browsers, and many others. Since the windowed application programs are separate and independent of the desktop, the multi-media content presented in the application program windows cannot effectively enhance the presentation on the desktop itself.

Further, multi-media content is made available in numerous formats. Still images are available in JPEG (Joint Photographic Experts Group), GIF, BMP (Windows®

bitmap), and other file formats. Sounds are distributed in WAV (wave), MIDI and other file formats. Video is distributed in MPEG (Motion Picture Experts Group), AVI and other file formats. The hypertext markup language (HTML) format is widely used to distribute documents or pages including text, images, video and sound on the World-Wide Web of the Internet. Three dimensional environments are now being developed in virtual reality markup language (VRML) and other formats. These various multi-media formats provide a facility for expressing multi-media content, but do not of themselves provide a facility for providing desktop functionality with multi-media enhancement.

The present invention provides multi-media content enhancements to the desktop of an operating system's graphical user interface. In one system according to the invention, an operating system shell synthesizes the display for the desktop into a hypertext multimedia document format (the HTML format, for example). The synthesized document includes the graphical icon oriented and menu driven user interface elements of the desktop, and also can include multi-media enhancements, such as text, graphics, sounds, animations, video, hypertext links, etc. These enhancements can add informative or explanatory content to the desktop, or otherwise customize the appearance and/or behavior of the desktop. The shell also acts as a hypertext multimedia document viewing software to display the synthesized document as the desktop in a graphical user interface, preferably as a full-screen background display to a windowing environment.

According to one aspect of the invention, the shell synthesizes the hypertext multimedia document from a template which contains the multi-media enhancements or references to the enhancements. The shell includes a pre-processor which processes the template and produces the synthesized document which is to be displayed as the desktop. This pre-processor converts soft or variable parameters into data in the hypertext multimedia document's format which is output in the synthesized document for display with the multi-media enhancements. The templates also contain document data for output in the synthesized document to cause embedding of a software object or objects in the displayed view of the synthesized document. These software objects implement the functionality of the desktop's graphical icon-oriented and/or menu driven user interface elements which control operating system and/or file system services. When the synthesized document is then displayed as the desktop in the graphical user interface, the object(s) provide the user interface features and functionality (e.g., for launching application programs, opening documents and files, drag and drop functionality, etc.) of the desktop. By embedding the objects in the synthesized document, these user interface features are displayed in combination with the multi-media enhancements by the shell.

According to a further aspect of the invention, the shell also synthesizes hypertext multimedia documents for display as the folder views and other displays in the shell's graphical user interface. The hypertext multimedia documents for the various displays are synthesized from templates which are identified in a configuration or ".ini" file. When the user navigates to one of the displays, the shell looks up the appropriate template to use for the display. The shell then processes the template into a hypertext multimedia document with embedded objects to provide the user interface elements required for the display (e.g., the graphical icons and drag and drop functionality in a folder view). The synthesized document is then displayed by the shell. The shell is thus able to provide multi-media content enhancements to these additional shell displays.

Synthesizing the desktop and other displays from templates further allows the multi-media enhancements to be easily and flexibly added and changed. The templates are in the format of the hypertext multimedia documents, and additionally contain directives to replace soft parameters by the pre-processor. The enhancements provided by the templates can be altered by editing the templates using hypertext document editing software or even a text editor. Alternatively, a new template or complete set of templates can be swapped in by changing entries in the configuration file. Accordingly, the overall appearance and behavior of the shell can be immediately changed by substituting the set of templates identified in the configuration files.

In an embodiment of the invention illustrated herein, the synthesized documents for the various displays are displayed in a single designated area or frame in the graphical user interface, such as the full-screen background display area of the windowing environment. As the user navigates from one display (e.g., the desktop) to another (e.g., a folder view), the shell switches the hypertext document displayed in the designated area in a fashion similar to navigating a hyperlink between hypertext documents.

The shell in the illustrated embodiment further operates as viewer or browser of hypertext documents. In addition to the synthesized hypertext documents for shell displays, the shell also displays and navigates between other hypertext documents, such as those available from the Internet. The shell thus extends hyperlink navigation and the rich multimedia content of hypertext documents to the shell's graphical user interface.

Additional features and advantages of the invention will be made apparent from the following detailed description of an illustrated embodiment which proceeds with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a computer system that may be used to implement a method and apparatus embodying the invention for incorporating multi-media enhancements to an operating system shell.

FIG. **2** is a data flow diagram showing the synthesis of a hypertext multimedia document by an operating system shell for a desktop display in a graphical user interface.

FIG. **3** is a view of a navigation bar incorporated by the shell of FIG. **2** in the folder display of FIG. **7**.

FIG. **4** is a block diagram of typical data structures for an object in the computer system of FIG. **1**.

FIG. **5** is an object framework in the shell of FIG. **2** which supports presenting a hypertext document incorporating graphical user interface functionality as a desktop display.

FIG. **6** is a view of a desktop display presented by the shell of FIG. **2**.

FIG. **7** is a view of a folder display presented by the shell of FIG. **2**.

DETAILED DESCRIPTION OF THE ILLUSTRATED EMBODIMENTS

1. Computer Overview

Referring to FIG. **1**, an operating environment for an illustrated embodiment of the present invention is a computer system **20** with a computer **22** that comprises at least one high speed processing unit (CPU) **24**, in conjunction with a memory system **26**, an input device **28**, and an output device 30. These elements are interconnected by at least one bus structure 32.

The illustrated CPU 24 is of familiar design and includes an ALU 34 for performing computations, a collection of registers 36 for temporary storage of data and instructions, and a control unit 38 for controlling operation of the system 20. The CPU 24 may having any of a variety of architectures including Alpha from Digital, MIPS from MIPS Technology, NEC, IDT, Siemens, and others, x86 from Intel and others, including Cyrix, AMD, and Nexgen, and the PowerPc from IBM and Motorola.

The memory system 26 generally includes high-speed main memory 40 in the form of a medium such as random access memory (RAM) and read only memory (ROM) semiconductor devices, and secondary storage 42 in the form of long term storage mediums such as floppy disks, hard disks, tape, CD-ROM, flash memory, etc. and other devices that store data using electrical, magnetic, optical or other recording media. The main memory 40 also can include video display memory for displaying images through a display device. Those skilled in the art will recognize that the memory 26 can comprise a variety of alternative components having a variety of storage capacities.

The input and output devices 28, 30 also are familiar. The input device 28 can comprise a keyboard, a mouse, a physical transducer (e.g., a microphone), etc. The output device 30 can comprise a display, a printer, a transducer (e.g., a speaker), etc. Some devices, such as a network interface or a modem, can be used as input and/or output devices.

As is familiar to those skilled in the art, the computer system 20 further includes an operating system and at least one application program. The operating system is the set of software which controls the computer system's operation and the allocation of resources. The application program is the set of software that performs a task desired by the user, using computer resources made available through the operating system. Both are resident in the illustrated memory system 26.

In accordance with the practices of persons skilled in the art of computer programming, the present invention is described below with reference to acts and symbolic representations of operations that are performed by computer system 20, unless indicated otherwise. Such acts and operations are sometimes referred to as being computer-executed. It will be appreciated that the acts and symbolically represented operations include the manipulation by the CPU 24 of electrical signals representing data bits which causes a resulting transformation or reduction of the electrical signal representation, and the maintenance of data bits at memory locations in memory system 26 to thereby reconfigure or otherwise alter the computer system's operation, as well as other processing of signals. The memory locations where data bits are maintained are physical locations that have particular electrical, magnetic, or optical properties corresponding to the data bits.

2. Shell Overview

With reference to FIG. 2, a shell 50 for an operating system of the computer 20 (FIG. 1) provides a graphical user interface for a user of the computer to interact with the operating system. The graphical user interface includes a desktop display 52 presented on a video screen of the computer's output device 30 (FIG. 1). The desktop display 52 preferably is presented by the shell in a windowing environment as a full-screen background display. Specifically, any visible windows that are not minimized to an icon are displayed overlaying the desktop display 52 in the windowing environment.

In the illustrated shell 50, the shell presents a variety of views 54 (listed in table 1 below) for different aspects of the graphical user interface on the desktop display 52. These views 54 are similar to displays presented in the graphical user interface of the Microsoft Windows® 95 operating system, but additionally include multi-media enhancements incorporated in the views in accordance with the invention. More particularly, the views 54 include the desktop and desktop container displays of the Microsoft Windows® 95 operating system. The desktop container displays are subdisplays of the Microsoft Window® 95 desktop, that contain a group of related graphical icons for file management or for launching application program and operating system services (e.g., My Computer, Network Neighborhood, Control Panel, etc.).

For each of the views 54 presented on the desktop display 52, the shell 50 synthesizes the view as a hypertext page 56. The illustrated hypertext page 56 is in a hypertext markup language (HTML) format. The HTML format is a standard format for multi-media hypertext documents which is used on the Word-Wide Web portion of the Internet. (The Internet is a global network of cooperatively interconnected computer networks, consisting of millions of individual computers. A portion of the Internet referred to as the "World-Wide Web" consists of computers, also referred to as "sites," which make multi-media documents in HTML format generally available for downloading or retrieval by individuals having a computer with Internet access.) HTML format documents are ASCII encoded files which employ tags to designate text formatting, hyperlinks, and multi-media content to be incorporated from other resources (e.g., files) into the document. Further details of the HTML format of the illustrated hypertext page 56 are described in the *HTML Reference,* attached hereto as Appendix Q.

A hypertext document is a document that contains hyperlinks. Hyperlinks are references to other documents which are generally indicated in a displayed view of the document as a graphic, underlined text, or highlighted text, and which can be activated by user input to the viewing software to cause the viewing software to navigate to the referenced document. A multimedia document is a document which contains or incorporates multiple forms and/or formats of information content such as text, images, sounds, software objects, video, animations, etc.

The shell 50 obtains the hypertext page 56 from which a view in the graphical user interface is synthesized from processing a stored hypertext template, or alternatively directly from a stored hypertext page. In the former case, the shell 50 includes a pre-processor 60 which synthesizes the hypertext page 56 from one or more of a set of templates 62 and one or more desktop interface controls 64. The templates 62 are files which contain data in the HTML format which is to be incorporated into the hypertext page 56, and additionally include pre-processor directives. The directives are instructions to the pre-processor for converting soft parameters into html formatted data in the hypertext page 56. The templates for each of the views in the illustrated shell are shown in the following Table 1.

TABLE 1

Hypertext View Templates

Friendly Name	File Name
Briefcase	brfcase.htm
Control Panel	control.htm
Default folder	folder.htm
Desktop	desktop.htm
Favorites folder	favorite.htm
File-system folder	directry.htm
My Computer	mycomp.htm
Network Neighborhood	nethood.htm
Printers	printer.htm
Recycle Bin	recycle.htm
Start Menu and subfolders	startmnu.htm
Workgroups	domain.htm
Vendor-specific workgroups	msdomain.htm
Vendor-specific networks	msnet.htm
Vendor-specific servers	msserver.htm
Servers	server.htm
Audio CD	audiocd.htm
Dial-Up Networking	dialupnt.htm
Entire Network	fullnet.htm
Fonts folder	fonts.htm
History	history.htm
My Documents	mydocs.htm
Network Workgroup	workgrp.htm
Program Files folder	progfile.htm
Root of data CD	datacd.htm
Root of floppy disk drive	floppy.htm
Root of hard disk	harddrv.htm
Windows folder	windows.htm

In general, the templates listed in Table 1 are related one-to-one with folders and are used to synthesize a hypertext page for a display associated with the related folders. Some of the folders correspond to actual directories in a file system of the computer's memory system **26** (FIG. 1). For example, each of the "windows folder," "root of hard disk," and "my documents" folders correspond to actual file system directories. The displays associated with these folder generally represent (at least in part) the contents of the corresponding directory, and are called "folder views." Others of the folders (termed "virtual folders") do not correspond to any file system directory. Accordingly, the displays associated with these folders generally do not represent the contents of a file system directory. For example, the "my computer" folder is a virtual folder.

The templates listed in Table 1 are for producing displays associated with a set of standard folders in the Windows® operating system. In addition to these standard folder templates listed in table 1, the templates **62** also can include additional templates for non-standard folders (herein called "custom templates"), such as folders corresponding to file system directories created by a user or added by a software installation program. For example, an installation program of an application software product (such as a productivity software, computer game, or utility software) that creates a new folder in which to install the application software's files also can add a template associated with the folder to the set of templates **62**. When the newly created folder is viewed in the graphical user interface, the shell **52** uses this added custom template to produce a folder view display representing the newly created folder's contents. These added custom templates can contain multimedia content enhancements specific to the new folder, such as graphic images, text, hyperlinks, or software objects relating to the application software product or its vendor.

The template (or alternatively stored hypertext page) to be used by the shell in synthesizing the hypertext page view in each display in the graphical user interface is identified in one or more configuration files **66**. The configuration files **66** can include both local and global configuration files. More particularly, folders that are actual file system directories can contain a hidden local configuration file (named "desktop.ini" in the illustrated computer **20**). (Hidden files are files having a flag or attribute which is set to indicate that the file normally is not displayed by a file management tool, e.g., the Windows Explorer in the Windows® 95 operating system.) This "desktop.ini" configuration files stores data identifying the template (e.g., by path and file name in the computer's file system) to be used in producing a folder view display of the folder. A listing of a representative desktop.ini file is shown in the following table 2.

TABLE 2

Representative Desktop.Ini File Listing.

```
[ExtShellFolderViews]
Default={FB7E5040-1F6D-11D0-89A9-00A0C9054129}
{FB7E5040-1F6D-11D0-89A9-00A0C9054129}={FB7E5040-1F6D-
11D0-89A9-00A0C9054129
}
{00000001-0001-0002-0003-000000000001}={25336920-03F9-11CF-
8FD0-00AA00686F13
}
{00000002-0001-0002-0003-000000000002}={00020900-0000-0000-
C000-000000000046
}
{00000003-0001-0002-0003-000000000003}={00020810-0000-0000-
C000-000000000046
}
[{00000001-0001-0002-0003-000000000001}]
PersistFile="pageone.html"
MenuName="friendly-name-for-view-1"
ToolTipText="Html View"
HelpText="This shows a HTML document"
[{00000002-0001-0002-0003-000000000002}]
PersistFile="word.doc"
MenuName="friendly-name-for-view-2"
ToolTipText="Word Document View"
HelpText="This shows a Word document"
[{00000003-0001-0002-0003-000000000003}]
PersistFile="Excel.xls"
MenuName="friendly-name-for-view-N"
ToolTipText="Excel Spreadsheet view"
HelpText="This shows an Excel spreadsheet"
[{FB7E5040-1F6D-11D0-89A9-00A0C9054129}]
IconArea_Image="c:\win95B\bubbles.bmp"
IconArea_Pos=1
```

The above representative desktop.ini file begins with a section having the heading "[ExtShellFolderViews]." This section lists globally unique identifiers ("GUIDs") associated with software objects that implement views of the folder in which the desktop. ini file is stored. A line beginning "default=" specifies a default view of the folder. The four lines below this specify alternative custom views of the folder in the format, <GUID>=<GUID>. The left GUID on each line identifies the software object that implements the view. If there is an entry with this left GUID in the system registry for the view object (i.e., the GUID is a CLSID registered in the system registry), then the line modifies some of the view's attributes. In such case, the right GUID is a CLSID that identifies the pre-processor **60** to be used for processing the template for that view, or alternatively identifies the viewer object (e.g., the HTML viewer **70** of FIG. **2** or other document object) which displays the hypertext page in the view (i.e., for use when the hypertext page itself is stored rather than a template from which the hypertext page is produced). If there is no entry with the left GUID in the system registry, then the line indicates a custom view and is unique only with the respective desktop.ini file.

More specifically, in the above representative desktop.ini file, the first or default line of the "[ExtShellFolderViews]" section identifies an object that implements a default view for the folder. The second line overrides some attributes of one of the folder's views. A separate section at the bottom of the desktop.ini file has lines which change the "IconArea_Image" and the "IconArea_Pos" attributes of that view, specifically the background bitmap of the view and its position (e.g., whether centered or tiled).

The third, fourth and fifth lines of the "[ExtShellFolderViews]" section specify custom views for the folder. The left hand identifiers are not CLSIDs of views registered in the system registry, and are unique only within the desktop.ini file. The right hand GUIDs on these lines are CLSIDs of objects registered in the system registry. Specifically, the right hand GUIDs are CLSIDs of the HTML viewer **70**, a Microsoft® Word document object, and a Microsoft® Excel document object, respectively. (Document objects are described below.) The desktop.ini file contains a section for each of the custom views which have the line "PersistFile= . . . " When one of these views is selected for display, the respective document object of the view is instantiated with the hypertext page, Word document, or Excel document, respectively, identified on the "PersistFile" line.

The template (or alternatively stored hypertext page) associated with a folder (whether an actual file system directory or virtual folder) also can be identified by entries in a global configuration file, which in the illustrated shell **52** is the system registry. In the Microsoft® Windows® operating system, the system registry is a database which stores configuration information for the operating system, including information to enumerate and track applications, device drivers, and operating system control parameters. For a detailed discussion of the registry, see Win32 Programmers Reference, Vol. 2, published by Microsoft Press, Redmond, Wash. (1993). Representative entries in the system registry for the illustrated shell **52** are listed in the following Table 3.

TABLE 3

Representative System Registry Entries for Folder Views

Folder Type	Registry Entry
Virtual Folder	HKCR\CLSID\{guid of virtual folder}\shellex\ExtShellFolderViews\{guid of view} PersistFile="template.htm"

Directory	HKCR\Directory\shellex\ExtShellFolderViews\{guid of view}
	PersistFile="directry.htm"

Default Folder	HKCR\Folder\shellex\ExtShellFolderViews\{guid of view}
	PersistFile="folder.htm"

Each of the representative registry entries in the above table include a GUID of a view which identifies either a pre-processor (where the view is synthesized from processing a template) or a document viewer (where the view is synthesized directly from a stored hypertext page or other document). If the GUID identifies a pre-processor (e.g., the pre-processor **60**) then the registry also contains an entry to identify the document viewer (e.g., the hypertext viewer **70** of FIG. **2**) for the hypertext page or other document produced from the pre-processor processing the template. This registry entry for the pre-processor has the form:

```
HKCR\CLSID\{guid of view}\FilterView="{guid of document
   viewer}"
```

These system registry entries that identify templates (or stored hypertext page or other document) to use in producing displays of the graphical user interface are created when the operating system shell **50** is first set up or installed. The registry entries can be edited manually or altered by software to substitute a different set of templates for the standard templates listed in Table 1 or to add to the standard templates.

The shell **50** determines the template (or stored hypertext page) to use in synthesizing the view of a particular folder from this information in the configuration files **66**. If the folder is an actual file system directory, the shell **50** checks for a desktop.ini configuration file in the directory. If the folder contains a desktop.ini configuration file which identifies a template (or stored hypertext page or other document), the shell **50** uses that template. If not, the shell **50** checks in the system registry for a registry entry under the "HKCR\directory" key (as shown in the representative registry entry for a directory in Table 3 above) that is specific to the folder and identifies its associated template (or stored document). If such a folder specific registry entry is found, the shell **50** uses the template identified in that folder-specific entry. If not, the shell **50** checks the system registry for a default folder entry under the "HKCR\folder" key (as shown in the representative default folder registry entry in Table 3 above). If the default folder entry is found, the shell **50** uses the template or document (e.g., the "folder.htm" template listed in the above Table 1) identified in the default folder entry. This default folder registry entry can be edited to change the default folder template (or document) used when no folder specific template is found.

If, on the other hand, the folder is a virtual folder, the shell **50** checks for an entry under the HKCR\CLSID key that lists a class identifier of the virtual folder (as shown in the representative registry entry for a virtual folder in Table 3 above). If such a virtual folder entry is found that identifies a template (or document), the shell **50** uses the identified template to produce the virtual folder's hypertext page **56**. Otherwise, the shell **50** looks to the default folder entry under the "HKCR\folder" key (as shown in Table 3 above), and uses the default folder template identified in that entry.

The particular folder for which the shell **50** synthesizes the current view **54** in the desktop display **52** is controlled by user action. At start-up, the shell synthesizes the desktop view from the "desktop.htm" template for the desktop folder. From this view, the user can navigate to other views (e.g., folder views of the my computer, network neighborhood, or file system directory folders) by activating icons or hyperlinks on the desktop display, or selecting commands from menus available on the current page. For example, the user can navigate to the my computer view by mouse clicking on the My Computer icon on the desktop view. Whereupon the shell **50** synthesizes the My Computer view from the "mycomp.htm" template, and displays the My Computer view.

With reference to FIG. **3**, a navigation toolbar **200** is presented by the shell **51** on folder view displays, such as the My Computer view. The user activates controls on the navigation bar to move between folder view displays, or back to the desktop display. The navigation toolbar **200** includes a drop down list control **202** in which the user can select to navigate to a desired folder. The destination folder can be selected from a list or typed into the control **202**. The navigation toolbar **200** also includes back and forward history controls **204** which the user can activate to move backwards and forwards through a history of previously navigated folders. The navigation toolbar **200** further includes a favorites menu button **206** which accesses a user specified list of folders. Each folder view display also may include other controls for navigating to another folder view display. The My Computer view, for example, contains icons which the user can activate to navigate to the floppy and hard disk drive folder views.

When synthesizing the hypertext page **56** for the current view, the pre-processor **60** parses the corresponding template(s) for the view (as shown in the Table 1 above or identified by the "desktop.ini" configuration file), and performs any pre-processor directives encountered in the templates. In the illustrated templates **62**, the pre-processor directives follow a syntax rule of being bracketed between the characters "<%" and "%>" to distinguish from HTML formatted data. The directives primarily specify soft parameters which the pre-processor converts into HTML formatted data for output into the hypertext page **56**. For example, the soft parameters can include the name of the folder whose contents are to be shown in a folder view, the name of the My Computer or other view (the user can rename these icons and corresponding views in the illustrated shell). Further details of the pre-processor directives are described below.

The pre-processor **60** utilized by the shell **50** for processing the templates **62** into hypertext pages **56** for use as displays in the graphical user inteface is identified by entries in the system registry. A class identifier ("CLSID") associated with the pre-processor is stored in the system registry under an "ExtShellFolderViews" key, and given a named value of "PersistFile=desktop.htm." The pre-processor **60** is implemented as an OLEICOM object (as described below). The pre-processor also has an associated entry under its CLSID in the system registry. This entry has a subkey "FilterView" under which is stored a CLSID associated with the document viewer (e.g., the HTML viewer **70** in the illustrated shell **50**). This allows a third party software vendor to substitute a custom pre-processor into the shell **50** in place of the illustrated pre-processor **60**.

The desktop interface controls **64** are software components (referred to as "objects") that implement the functionality of a particular aspect of the shell's graphical user interface. For example, the desktop interface control for the desktop view provides the graphical icons (e.g., My Computer, Network Neighborhood, and Recycle Bin icons), menus, and functionality (e.g., drag and drop, icon and menu activation, and other operations) of that view. The control which is incorporated into the hypertext page **56** for the folder views provides the large icon, small icon, list and detail mode graphical icon oriented user interface and functionality of the folder views in the Microsoft Windows® 95 operating system. These controls include interfaces which allow the controls to interoperate with the shell to provide the view on the desktop display **52**, and with the operating system to provide file management and to launch application programs and operating system services. The desktop interface controls **64** are embedded in the hypertext page **56** using HTML tags as described in further detail below.

The illustrated shell **50** also includes a hypertext viewer **70** and a shell explorer **72** which provide the desktop display **52** on the video screen of the computer's output device **30** (FIG. 1). The hypertext viewer **70** is a software component in the shell **50** which operates to parse the HTML formatted data in the hypertext page **56**, and produce the view **54** of the hypertext page for display in the desktop display **52**. In effect, the hypertext viewer **70** contains the HTML parsing and display code equivalent to an Internet browser. The illustrated hypertext viewer supports parsing and display of the same HTML format which is supported in the Microsoft® Internet Explorer.

The shell explorer **72** is another software component of the shell which manages graphical user interface elements of the desktop display other than the view. More specifically, the shell explorer **72** provides a frame of the desktop display having a display area in which the hypertext viewer **70** draws the view **54** of the hypertext page **56**. The frame includes user interface elements (e.g., the task bar **78** with start button and menu) that continue to be displayed on the desktop display when the hypertext view changes. In effect, the shell explorer **72** acts as a host or container of the hypertext page view **54** displayed by the hypertext viewer **70**. The shell explorer **72** and hypertext viewer **70** cooperate to embed the view **54** provided by the hypertext viewer **70** in the desktop display's frame provided by the shell explorer **72**.

3. Object Overview

With reference to FIG. 3, the shell explorer **72** and the hypertext viewer **70** are implemented as objects which conform to Microsoft Corporation's Component Object Model (COM), and support various ActiveX (also known as Object Linking and Embedding ("OLE")) interfaces. COM, ActiveX and OLE are object-oriented technologies which provide integration and interoperahility between separate software components. For a detailed discussion of OLE see *Inside OLE, Second Edition* by Kraig Brockschmidt, Microsoft Press, Redmond, Wash. 1995. A brief overview of objects in OLE and associated terminology is provided below before discussing the details of the illustrated embodiment.

Using these object-oriented technologies, the illustrated shell **50** (FIG. 2) works with the hypertext page **56** by encapsulating the hypertext page into an associated object (i.e., the hypertext viewer **70**), and integrating with the object using pre-defined interfaces (hereafter referred to as the document object interfaces, and also referred to as the IOLEXXX/IMSOXXX interfaces, which are described in more detail in the *OLE Document Objects Specifications*, attached hereto as Appendix R). The hypertext viewer document object includes code to work with the hypertext page **56** (i.e., data in the HTML format), including code to display a view of the bypertext page in the desktop display **52**. The document object interfaces of the hypertext viewer allow integration with the shell explorer **72**, which operates as a host or container of document objects (i.e., by providing the desktop display **52** having a display area in which a hosted document object—the hypertext viewer **70**—can display its view of the hypertext page **56**). As a document object host, the shell explorer **72**, can host any variety of document (e.g., documents associated with the Microsoft Word, Microsoft Excel, and other application programs from Microsoft or other software developers) which is encapsulated by an object that supports the document object integration interfaces. Implementing the hypertext viewer as a document object also allows the desktop and folder views of the synthesized hypertext page **45** to be embedded in other document object containers, such as a file manager, Internet browser or other application program which is implemented as a document object container. An example in which a view of the hypertext page **56** produced by the hypertext viewer **70** is hosted in the desktop display **150** of the shell explorer **72** is shown in FIG. **5** and described in more detail below. A further example in which a view of the hypertext page **56** produced by the hypertext viewer **70** is hosted in a window of an application is shown in FIG. **6** and described in more detail below.

An object is an instance of a programmer-defined type referred to as a class, which exhibits the characteristics of data encapsulation, polymorphism and inheritance. Data encapsulation refers to the combining of data (also referred to as properties of an object) with methods that operate on the data (also referred to as member functions of an object) into a unitary software component (i.e., the object), such that the object hides its internal composition, structure and operation and exposes its functionality to client programs that utilize the object only through one or more interfaces. An interface of the object is a group of semantically related member functions of the object. In other words, the client programs do not access the object's data directly, but must instead call functions on the object's interfaces to operate on the data.

Polymorphism refers to the ability to view (i.e., interact with) two similar objects through a common interface, thereby eliminating the need to differentiate between two objects. Inheritance refers to the derivation of different classes of objects from a base class, where the derived classes inherit the properties and characteristics of the base class (which for purposes of OLE are the interfaces of the base class).

Microsoft Corporations's COM specification defines binary standards for objects and their interfaces which facilitate the integration of software components. According to the COM specification, a typical object **80** is represented in the computer system **20** (FIG. **1**) by an instance data structure **82**, a virtual function table **84**, and member functions **86–88**. The instance data structure **82** contains a pointer **90** to the virtual function table **84** and data **92** (also referred to as data members, or properties of the object). A pointer is a data value that holds the address of an item in memory. The virtual function table **84** contains entries **96–98** for the member functions **86–88**. Each of the entries **96–98** contains a reference to the code **86–88** that implements the corresponding member function.

The pointer **90**, the virtual function table **84**, and the member functions **86–88** implement an interface of the object **80**. Client programs interact with the object **80** by obtaining a pointer (referred to as an interface pointer) to the pointer **90** of the virtual function table **84**. OLE includes a type definition of an interface pointer which allows client programs to call member functions on the interface by name through the interface pointer and provides type checking on the function's arguments, as expressed in the following code (in the C++programming language):

```
pInterface->MemberFunction( . . . )
```

By convention, the interfaces of an object are illustrated graphically as a plug-in jack as shown for the document object in FIG. **3**. Also, Interfaces conventionally are given names beginning with a capital "I." Objects can include multiple interfaces which are implemented with one or more virtual function tables. The member function of an interface is denoted as "IInterfaceName::FunctionName." The object **80** conforming to the COM specification exhibits data encapsulation by exposing its interfaces (semantic groupings of its member functions) to client programs. The client programs interact with the object **80** by calling the member functions **86–88** on a particular interface of the object, but do not directly manipulate the object's data. The object **80** also exhibits polymorphism and inheritance in that the object **80** can provide interfaces in common with a base class and other similar objects, so that client programs can interact with each of the objects in the same manner by calling member functions of the interface that the objects have in common.

4. Document Object and Server Overview

Referring still to FIG. **3**, the virtual function table **84** and member functions **86–88** of the object **80** are provided by a server application program **100** which is stored in the computer system **20** (Fig. **1**) as an executable program file (with a ".exe" file name extension) or as a dynamic link library file (with a ".dll" file name extension). Dynamic link library files are loaded, dynamically linked, and executed by the Windows® operating system in a same process with a client application program. Executable program files are loaded by the operating system as a separately executing process. In accordance with OLE, the server application **100** includes code for the virtual function table **84** (FIG. **3**) and member functions **86–88** (FIG. **3**) of the classes that it supports, and also includes a class factory **102** that generates the instance data structure **82** (FIG. **3**) for an object of the class.

A server application can be written by a programmer to support a particular class of object that contains any desired data. More specifically, a programmer can write server applications which provide objects that contain the data of a particular variety of computer document (e.g., document **56** of FIG. **2**), such as a text document, spreadsheet, drawing, etc., or that contain data for part of a computer document, such as a range of spreadsheet cells, a paragraph of a text document, etc. These objects which contain document data (and additionally support the document object interfaces described above) are referred to herein as document objects. (Further details of document object are described in the *OLE Document Objects Specifications,* attached hereto as Appendix R.) For example, software application programs such as Microsoft® Word can be written as a server application in which the application program's documents are represented as OLE objects. The illustrated shell **50** (FIG. **2**) includes a server application which represents the hypertext page **56** as a document object (i.e., the hypertext viewer **70**). This allows the shell explorer **72** (FIG. **2**) and other document object containers to interact with the hypertext page **56** through the document object interfaces.

For a client program (e.g., a document object host or container) to interact with the object **80** provided by the server application **100**, the server application must first create the object (i.e., instantiate an object of a class supported by the server application) and the client program must gain an interface pointer to the object **80**. In OLE, the client program realizes these events using services provided by OLE and a set of standard object interfaces defined by COM based on class and interface identifiers assigned to the object's class and interfaces. More specifically, the services are available to client programs as application programming interface (API) functions provided in the COM library, which is part of a component of the Windows® operating system in a file named "OLE32.DLL." In OLE, classes of objects are uniquely associated with class identifiers ("CLSIDs"). Class identifiers are 128-bit globally unique identifiers ("GUID") that the programmer creates with an OLE service named "CoCreateGUID" and assigns to the respective classes. The interfaces of an object are associated with interface identifiers ("IIDs").

In particular, the COM library provides an API function, "CoCreateInstance," that the client program can call to request creation of an object to encapsulate a particular documents data using a CLSID associated with the data. The CoCreateInstance API function creates the object and returns a pointer of the requested interface to the client program.

Once the client program has obtained a first interface pointer to the object **80**, the client program obtains pointers to other desired interfaces of the object using the interface identifier associated with the desired interface. COM defines several standard interfaces generally supported by OLE objects including the IUnknown interface. This interface includes a member function named "QueryInterface." The QueryInterface function can be called with an interface identifier as an argument, and returns a pointer to the interface associated with that interface identifier. By convention, the IUnknown interface's member functions are included as part of each interface on an object. Thus, any interface pointer that the client program obtains to an interface of the object **80** can be used to call the QueryInterface function.

In a typical situation, however, the only information that the client program has to reference a particular document is a textual name, such as a file name or an Internet URL. In the case of a file name, the COM library provides API functions (e.g., "GetClassFile," "ReadClassStg" and "ReadClassStm") for obtaining a CLSID associated with the file. Client programs also can utilize a system provided object known as a moniker to resolve a name that references a document into an interface pointer on an instantiated object that encapsulates the document's data. These well known mechanisms are described in more detail in *Inside OLE, Second Edition,* supra.

5. Shell Objects and Interfaces

In addition to the document object interfaces, the illustrated shell **50** (FIG. **2**) also utilizes an object-oriented framework of the Windows Explorer application, folder view windows, and "File Open" dialog in the Microsoft Windows® 95 operating system. This framework defines three interfaces, a shell browser ("IShellBrowser") interface **122**, a shell view ("IShellView") interface **126** and a shell folder ("IShellFolder") interface **128**, which are described in further detail in the Appendix S attached hereto. The shell browser interface **122** is supported on an object which acts as a container for views of a namespace, such as the folders in a file system. This container object provides a top level window having a frame in which to embed the namespace view. Examples of such a container object include the object which provides the top level or outer window in the Windows Explorer application, and also the object which provides the top level window for folder views in the shell of the Windows® 95 operating system. The namespace is a collection of symbols (e.g., names) associated with items managed through the container object, and rules for resolving a given name into the item it designates. In the case of an operating system shell, the namespace may include file names, directory names, storage devices, registry keys, printers, network resources, etc.

The shell browser interface **122** allows the namespace view contained in such a container window to insert menu items with those of the container window into a composite menu, install the composite menu into the container's window, and also remove the container's menu items from the composite menu. The shell browser interface **122** also allows the namespace view to set and display status text relevant to the view in the container's window. The view also can enable and disable the container's modeless dialog boxes, and translate accelerator keystrokes intended for the container through functions of the container object exposed by the sbell browser interface **122**.

The shell view interface **126** is supported on a view object which provides a namespace view for display in the container object's window. In the Windows® 95 operating system, for example, includes a default view ("DefView") object **124** which provides a view of the desktop in the Windows® 95 shell's graphical user interface which graphically represents the desktop, including the graphical icon-oriented and menu user interface elements of the desktop view (i.e., the full screen background display with the My Computer, Network Neighborhood, and Recycle Bin icons, drag and drop functionality, and pop-up menus of the desktop). Other view objects in the Windows® 95 operating system provide the large icon, small icon, list and details views representing a folder's contents which are displayed in the folder view windows and the Windows Explorer application's window. The container object for these windows communicate with the view objects through the shell view interface **126**. The communication involves the translation of window messages, the state of the container object's window (e.g., activated or deactivated), the merging of menus, and tool bar items.

The shell folder interface **128** is provided in this framework for implementing on objects that manage or extend the namespace. The shell folder interface **128** provides functions to display or operate on the contents of the namespace. For example, an object supporting the shell folder interface can add additional groups of symbols to the namespace, such as uniform resource locators ("URLs") of World-Wide Web pages.

In the illustrated shell **50**, the shell browser interface **122** is implemented on a shell explorer object **120** which provides the desktop display **52** (FIG. **2**) and acts as a container for the hypertext view **54**. As described above, the desktop display **52** operates as the desktop of a windowing environment of a graphical user interface provided by the shell **50**. More particularly, the shell explorer object **120** in the illustrated shell **50** is the Shell Explorer from the Microsoft Windows® 95 operating system which manages the Windows® 95 operating system's overall graphical user interface and windowing environment, and acts as a container for the desktop and folder views provided by the default view object **124**.

The illustrated shell **50** (FIG. **2**) extends the framework of the Windows® 95 operating system shell to also support document object hosting by the shell explorer **120**, so as to allow displaying the view **54** of the synthesized hypertext page **56** in the desktop display **52**. This is achieved in the illustrated shell **50** by a web view object **130** which supports the document object container interfaces **132** in addition to the shell view interface **126** and the shell folder interface **128**. By supporting these interfaces, the web view object **130** effectively operates as a bi-directional proxy object. The web view object **130** can act as both a shel view object that can by hosted by the shell explorer **120**, and as a document object container that can host the hypertext viewer **70** or other document objects. Since the web view object **130** supports the shell view and shell folder interfaces, it can take the place of the default view object **124** in the framework of the Windows® 95 operating system shell.

6. Desktop Interface Controls

With reference still to FIG. **4**, the default view object **124** (FIG. **2**) from the Windows® 95 shell framework is modified in the illustrated shell **50** for embedding in the hypertext page **56** as one of the desktop interface controls **64**. More specifically, the desktop interface controls are implemented as ActiveX controls. An ActiveX control is an object which conforms to the COM specification, and may support additional OLE interfaces such as those for drag and drop operations and other OLE functions. In particular, the desktop interface controls **64** (including the default view object **124**) support OLE interfaces that allow the object to be embedded in a document object. In other words, interfaces which allow a document object such as the hypertext viewer **70** to act as a container or host of the embedded desktop interface controls **64**. Further details of ActiveX controls are described in in the *OLE Control and Control Container Guidelines,* attached hereto as Appendix T.

In the illustrated shell **50** (FIG. **2**), the desktop interface controls **60** are embedded in the hypertext page **56** using an HTML object embedding tag, which has the following format:

```
<OBJECT classid=codebase=data=height=width=>
```

The classid parameter of this tag (if present) specifies a class identifier of the control. The hypertext viewer uses the class identifier to create the control, such as by calling the CoCreateInstance API function to cause the control's server application to be loaded and the server application's class factory to instantiate the control. The codebase parameter (if present) specifies a URL of a file containing the control (such as on the Internet). If the control is not installed on the computer **20**, the hypertext viewer can retrieve this file using the URL from the Internet and then install the file on the computer before instant control using its class identifier. The data tag (if present) specifies persistent data for the control as either a text string or via a URL of a file containing the control's persistent data. Further details of the HTML object embedding tag are described in the *HTML Reference,* attached hereto as Appendix Q.

The templates **62** which are processed into the hypertext page **56** include the HTML object embedding tags which specify the desktop interface controls **64** to embed in the displayed hypertext view **54**. Since the HTML object embedding tags already are in the HTML format, the pre-processor **60** outputs the tags along with other HTML format data from the templates **62** directly into the hypertext page **56** without conversion.

When displaying the hypertext view **54** of the hypertext page **56**, the hypertext viewer **70** parses the HTML object embedding tags along with the other HTML format data in the hypertext documents. On encountering the HTML object embedding tags for the desktop interface controls **64** during the parsing, the hypertext viewer **70** instantiates the desktop controls **64** using the class identifiers specified in the tags. If the server applications for the controls **64** are not installed on the computer **20** (FIG. **1**), the hypertext viewer **70** can download the server application using the URLs specified as the codebase attribute of the tags (if any). The hypertext viewer **70** then displays the instantiated desktop interface controls **64** together with the other content of the hypertext page in the hypertext view **54**. In addition to the desktop interface controls **64** which implement the desktop and folder view functionality of the shell, other ActiveX controls also can be incorporated in the same manner to provide further multi-media enhancements in the form of executable software content.

7. Template Pre-Processor Directives

As described above with reference to FIG. **2**, the templates **62** contain directives which control processing of the templates by the pre-processor **60** into the hypertext page **56**. The directives can cause the pre-processor to perform several different operations, which include converting parameters with variable values into HTML format data for output into the hypertext page **56**, processing additional templates into sub-pages of the hypertext page **56**, and also specifying a particular pre-processor to use for the pre-processing. As described above, the directives each begin with the characters "<%" and end with the characters "%>" to distinguish from the HTML data in the templates **62**.

Each of the illustrated templates **62** begin with the following initial directive, which the pre-processor **60** uses to detect whether the template is a valid template and to detect its beginning.

```
<%WinShell=CreateObject("WinShellHTMLPreProc")%>
```

As this initial directive indicates the beginning of the template, the pre-processor **60** processes only directives that appear after this in directive in the template and ignores any directive appearing on any lines before the initial directive in the template. If this initial directive is not encountered in the template, the template is considered invalid.

The later directives in each template specify parameters whose value the pre-processor converts into an HTML format text string for output into the hypertext page **56**. In the illustrated templates **62**, these directives have the format of the text string "WinShell." followed by a tag which designates the particular parameter to be converted, as shown in the following example directives:

```
<%WinShell.Title%>
<%WinShell.TemplateDirPath%>
<%WinShell.ProcessFile("dsk_1.htm")%>
<%WinShell.IfProxy("http://iptdweb.microsoft.com/activedeskt
op/desktop/tickhost.htm","%WINDIR%\web\ticker.htm")%>
```

For some of these directives, the conversion of parameters into HTML format text strings by the pre-processor **60** is as simple as outputting a current value of the parameter in the form of a text string. For example, the illustrated pre-processor **60** responds to the "<%WinShell.Title%>" and the "<%WinShell.TemplateDirPath%>" directives by simply outputting a text string of the designated parameter's current value. The tag "Title" refers to a parameter whose value is a name of the view which is produced from the template, e.g., "Desktop" for the desktop view, or the name of a particular folder for a folder view. The tag "TemplateDirPath" refers to a directory path name of the directory in the file system where the templates **62** are stored.

Other directives require move complex processing by the pre-processor **60** for conversion to an HTML format text string. The "<%WinShell.ProcessFile("dsk_1.htm")%>" directive for example, causes the pre-processor **60** to also process another of the templates **62** (e.g., the "dsk_1.htm" file specified in the tag) into an HTML format file, then output the name of that file in the form of an HTML text string as the value of the parameter. As another example, the directive with the tag "IfProxy" instructs the pre-processor to output a first text string if the shell **50** is running on a computer which uses a proxy server on a local area network for connecting to the Internet, and to output a second text string otherwise.

8. Desktop Hypertext Page View Example

FIG. **5** illustrates an example hypertext desktop view **150** produced and displayed in the desktop display **52** (FIG. **2**) on a video screen of the computer **20** (FIG. **1**) according to the illustrated embodiment of the invention. The example hypertext desktop view **150** is produced in the illustrated shell **50** by processing a set of templates using the pre-processor **60** (FIG. **2**), including a "Desktop.htm" template, and a "Dsk_1.htm" template which are listed in the attached Appendices A and B, respectively. As a result of this processing, the pre-processor **60** outputs a hypertext page consisting of a "Sfv2395.tmp" and a "Sfv15143.htm" HTML format files which are listed in the attached Appendices G and H, respectively. This synthesized hypertext page additionally incorporates data from the HTML format files, "Infopane.htm," "news.htm," "ticker.htm," and "tickhost.htm" which are listed in the attached Appendices C through F, respectively. The synthesized hypertext page is parsed by the hypertext viewer **70** to generate the example hypertext desktop view **150**.

The example hypertext desktop view **150** integrates a variety of multi-media elements with the graphical icon oriented and menu driven user interface elements of the desktop. The example hypertext desktop view **150** has separate HTML frames (i.e., areas on an HTML page) for a ticker **152**, a news pane **154**, and desktop icons frame **156**. The ticker **152** presents scrolling HTML format information retrieved from the Internet, such as stock quotes and sports scores. The news pane **154** also presents HTML format information retrieved from the web, such as images and text relating to news events. The desktop icons frame **156** includes the desktop interface control **64** (FIG. **2**) which provides the graphical icon oriented and menu driven user interface elements of the desktop, including the "My Computer," "Network Neighborhood," "Recycle Bin" icons and other user installed icons as well as the drag and drop and other operations of the desktop user interface as in the Windows® 95 operating system shell.

9. Folder Hypertext Page View Example

FIG. **6** illustrates an example folder view **170** produced and displayed within an Explorer window **172** on a video screen of the computer **20** (FIG. **1**) according to the illustrated embodiment of the invention. The example folder view **170** is produced in the illustrated shell **50** by processing a set of templates using the pre-processor **60** (FIG. **2**), including a "Directory.htm" template, a "Dir__1.htm" template, a "Dir__2.htm" template, and a "Dir__3.htm" template which are listed in the attached Appendices I through L, respectively. As a result of this processing, the pre-processor **60** outputs HTML format files, "Sfv6190.tmp," "Sfv67916.htm," Sfv47917.htm," and "Sfv47945.htm." The "Sfv6190.tmp" HTML file is a synthesized hypertext page which incorporates HTML format data from the other files. The synthesized hypertext page is parsed by the hypertext viewer **70** to generate the example folder view **150**.

The example hypertext folder view **170** is shown hosted in an Explorer application window **172** in FIG. **6**, rather than the desktop display **52** (FIG. **2**) as was the desktop view **150** (FIG. **5**). The Explorer application in this example is implemented using the object-oriented framework shown in FIG. **4**. However, in this example, the shell explorer object **72** provides an application window similar to that of the Windows Explorer application in the Windows® 95 operating system. Using the framework, the Explorer application hosts the hypertext viewer **70** as a document object which allows the view **54** of the hypertext page **56** to be displayed in Explorer application's window **172**.

The example hypertext folder view **170** integrates a variety of multi-media elements with the graphical icon oriented and menu driven user interface elements of Windows® 95 folder views (i.e., the large icon, small icon, list and detail folder views). The example hypertext folder view **150** represents the contents of the folder, "c:\dos," on the computer **20** (FIG. **1**), and has separate HTML frames for a title banner **176**, a supplemental hypertext frame **178**, and a folder icons frame **180**. The title banner **176** presents a title (specifically a name of the folder represented by the folder view) with formatting (e.g., font face, size, etc.) as specified by the HTML data in the hypertext page, and hypertext links which the user can activate to navigate to further hypertext views (i.e., alternative hypertext views of the folder, and a hypertext help page with instructions on operating the Explorer application). The supplemental hypertext frame **178** includes text and hypertext links to supplement the folder's contents. The user can navigate these hypertext links to hypertext pages on the Internet. The folder icons frame **180** includes an embedded desktop interface control **64** which provides the graphical icon oriented and menu driven user interface elements of the folder view, including icons representing the files and sub-folders contained in the folder (i.e., the "c:\dos" folder) represented in the folder view as well as the icon drag and drop, scrolling and other user interface operations as are available in the folder views of the Windows® 95 operating system.

The example hypertext desktop view **150** and example hypertext folder view **170** show a few of the multi-media elements that can be incorporated with graphical user interface elements into the hypertext view **54** for display in the desktop display **52** or an Explorer application window. In the illustrated shell **50**, the hypertext viewer **70** contains code to parse and display hypertext pages in the HTML format, including any of the HTML tags supported by the Microsoft® Internet Explorer 3.0 browser software. The hypertext views in the illustrated shell therefore can incorporate any of the multi-media elements implemented using these HTML tags, including HTML frames, tables, fonts, hypertext links, embedded images, embedded sounds, embedded objects, and others. Other multi-media elements also can be incorporated in the hypertext views by modifying the hypertext parsing and display code of the hypertext viewer **70** to also support appropriate tags for such elements.

In an alternative embodiment of the invention, the shell **50** can produce displays in the graphical user interface directly from stored hypertext pages. More particularly, in place of a stored set of templates **62** (FIG. **2**), the hypertext page for each display is stored and identified in the configuration files **66**. This elimates pre-processing of a stored template to produce the hypertext page whenever a display is generated, and thus speeds generation of displays in the graphical user interface. The shell **50** therefore omits the pre-processor **60** in this alternative embodiment.

In some alternative embodiments of the invention, multimedia document formats other than HTML can be used for the pages **56** (FIG. **2**). Further, the format of the pages **56** can be other than a hypertext format, or in other words need not allow for hyperlinks to be incorporated into the pages **56**. For example, a multimedia document format which does not include hyperlinks, such as the document formats used in the Microsoft® Word and Microsoft® PowerPoint application software, can be used.

Having described and illustrated the principles of our invention with reference to an illustrated embodiment, it will be recognized that the illustrated embodiment can be modified in arrangement and detail without departing from such principles. It should be understood that the programs, processes, or methods described herein are not related or limited to any particular type of computer apparatus, unless indicated otherwise. Various types of general purpose or specialized computer apparatus may be used with or perform operations in accordance with the teachings described herein. Elements of the illustrated embodiment shown in software may be implemented in hardware and vice versa.

In view of the many possible embodiments to which the principles of our invention may be applied, it should be recognized that the detailed embodiments are illustrative only and should not be taken as limiting the scope of our invention. Rather, we claim as our invention all such embodiments as may come within the scope and spirit of the following claims and equivalents thereto.

We claim:

1. In a computer having a display device, a system for displaying a user interface to an operating system as a hypertext multimedia document on the display device, comprising:

a graphical user interface for providing a windowing environment supporting a plurality of windows displayed on the display device according to a front to back order wherein a window towards the front in the order overlaps any windows farther back in the order which are displayed in a same area of the display device; and

a desktop in the windowing environment of the graphical user interface for providing a view of a hypertext multimedia document as a full-screen graphical background display over which the windows are displayed.

2. The system of claim **1** comprising:

a plurality of icons in the hypertext multimedia document and operative to launch application programs on user activation.

3. The system of claim **1** comprising:

a plurality of icons in the hypertext multimedia document and operative to activate file system services.

4. The system of claim **1** comprising:

a representation in the hypertext multimedia document of the contents of a folder in a file system of the computer.

5. An operating system shell comprising:

a hypertext multimedia document template;

a preprocessor for processing the template into a hypertext multimedia document having an embedded control object for providing at least part of an operating system's graphical user interface; and

a viewer for displaying a view of the hypertext document in the graphical user interface wherein the embedded control object is manipulable by a user to activate a service of the operating system.

6. The operating system shell of claim **5** comprising:

a view container object for hosting the viewer and providing a desktop display in which the view is embedded as a full-screen graphical background display of a windowing environment.

7. The operating system shell of claim **5** comprising:

a view container object for hosting the viewer and providing an application window in which the view is embedded.

8. A template for use in a system to synthesize a hypertext multimedia document to provide an operating system user interface, the system having a preprocessor, an embeddable user interface control object, and a hypertext multimedia document viewer, the template comprising:

hypertext data for incorporating multi-media information into the hypertext multimedia document;

at least one preprocessor directive for converting parameters into hypertext data; and

an object insertion tag for embedding the user interface control object into the hypertext multimedia document, wherein the embedded control object is manipulable by a user to activate a service of the operating system, whereby the preprocessor processes the template into a hypertext multimedia document which provides an operating system user interface when displayed by the viewer.

9. A method of providing a graphical user interface to an operating system of a computer having a video screen, comprising:

providing an operating system user interface control manipulable by a user to activate a service of the operating system;

providing a hypertext page having hypertext data for incorporating multi-media enhancements, and an object insertion tag indicative of said control for embedding said control;

generating a view of the hypertext page incorporating said multi-media enhancements and with said control embedded therein; and

displaying the view on the video screen.

10. The method of claim **9** comprising:

displaying the view in a desktop display of a windowing environment on the video screen, the desktop display being a full-screen graphical background display of the windowing environment.

11. The method of claim **9** comprising:

displaying the view in a window of a windowing environment on the video screen.

12. The method of claim **9** comprising:

providing a template having the hypertext data, the object insertion tag, and a pre-processor directive specifying a parameter; and

processing the template to synthesize the hypertext page from the template, the processing including converting the parameter into hypertext data.

13. The method of claim **12** wherein the view of the hypertext page represents a folder of a file system managed by the operating system, the method comprising:

storing a configuration file in the folder specifying the template; and

when a user navigates to the folder, identifying the template to use for synthesizing the hypertext page from the configuration file.

14. A file system navigation method comprising:

providing a plurality of templates associated with folders in a file system, the templates having an object insertion tag for embedding a folder user interface control;

when a folder is opened by a user, performing the steps of:

processing the template associated with the opened folder to produce a hypertext page having the object insertion tag; and

displaying a view of the hypertext page with the folder user interface control embedded therein, whereby the user manipulates the folder user interface control to activate file system services relating to the folder.

15. The method of claim **14** comprising:

storing configuration files in at least some of the folders, each configuration file containing data identifying the template associated with the respective folder in which the configuration file is stored;

if the opened folder stores one of the configuration files, determining the template associated with the opened folder from the configuration file stored in the opened folder; and

otherwise, determining a default template is associated with the opened folder.

16. A hypertext page, comprising:

hypertext data; and

an object insertion tag for embedding a user interface control object, said object providing graphical icon-oriented and/or menu driven user interface elements for controlling operating system and/or file system services when the hypertext page is viewed.

17. A desktop provided by an operating system shell on a computer as a full-screen graphical background display of a hypertext multimedia document in a windowing enviroment of a graphical user interface, comprising:

a plurality of graphical icon-oriented and menu-driven user interface elements provided by a control object, the user interface elements being manipulable by a user of the computer to initiate operating system services; and

a view of a hypertext multimedia document incorporating multi-media enhancements and having the control object embedded therein for integrating the multi-media enhancements with the user interface elements, the view constituting at least part of the full screen graphical background display of the desktop in the windowing environment.

* * * * *

PATENT APPLICATION SERIAL NO. 08/761699

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

PTO-1556
(5/87)

08/761,699

OPERATING SYSTEM SHELL WITH HYPERTEXT DESKTOP

ABSTRACT OF THE DISCLOSURE

An operating system shell provides a graphical user interface having a windowing environment with a desktop. The shell synthesizes a hypertext page for display as the desktop in the graphical user interface. The hypertext page has an embedded software object which provides graphical icon-oriented and menu-driven user interface elements for activating operating system services in the displayed hypertext page. The shell also provides windowed hypertext pages for managing file system folders. The shell synthesizes the hypertext pages from templates which can be edited to incorporate a variety of multi-media enhancements with the user interface elements in the graphical user interface. Templates can be associated with specific folders in the file system to provide folder specific hypertext pages integrated with user interface elements for managing the folder.

#7

1090.00-16.

OPERATING SYSTEM SHELL WITH HYPERTEXT DESKTOP

FIELD OF THE INVENTION

This invention relates to a user interface or shell of an operating system, and more particularly relates to the incorporation of rich and dynamic multimedia content into such an interface.

BACKGROUND AND SUMMARY OF THE INVENTION

It is now common for operating systems to have a shell which provides a graphical user interface (GUI). The shell is a piece of software (either a separate program or component part of the operating system) that provides direct communication between the user and the operating system. The graphical user interface typically provides a graphical icon-oriented and/or menu driven environment for the user to interact with the operating system.

The graphical user interface of many operating system shells is based on a desktop metaphor. More specifically, the graphical user interface is intended to create a graphical environment which simulates working at a desk. These graphical user interfaces typically employ a windowing environment with a desktop.

The windowing environment presents the user with specially delineated areas of the screen called windows, each of which is dedicated to a particular application program, file or document. Each window can act independently, as if it were a virtual display device under control of its particular application program. Windows can typically be resized, moved around the display, and stacked so as to overlay another. In some windowing environments, windows can be minimized to an icon or increased to a full-screen display. Usually, the windows have a top to bottom order in which they are displayed, with top windows at a particular location on the screen overlaying any other window at that same location. The top-most window has the "focus" and accepts the user's input. The user can switch other windows to the top by clicking with a mouse or other pointer device, or by inputting certain key combinations. This allows the user to work with multiple application

2

programs, files and documents in a similar manner as physically working with multiple paper documents and items which can be arbitrarily stacked or arranged on an actual desk.

The desktop of the graphical user interface is a screen display containing icons representing programs, files and resources available to the user. As such, the desktop acts as a launching point for running application programs, opening documents or files, and initiating operating system services. In accordance with the desktop metaphor, the desktop simulates the top of an actual desk on which various work items are made available to the office worker. The desktop in some graphical user interfaces thus includes icons representing resources found on a real desk, such as a file cabinet, telephone, wastebasket, and scratchpad, which are used to access their computer equivalents. In typical graphical user interfaces, the desktop always remains as a full-screen background display relative to the windowing environment. In other words, the desktop cannot be moved or resized, and all visible windows of the windowing environment overlay the desktop as would paper documents and other items on top of an actual desk.

A drawback to many prior desktops is their limited capacity to present multi-media content enhancements. For example, the shell of the Microsoft Windows® 95 operating system provides a graphical user interface with a windowing environment and a desktop. As a default, this desktop includes a "my computer" icon, a "network neighborhood" icon, and a "recycle bin" icon against a solid color background, and also includes a task bar along a bottom edge of the screen with a "start" button for menu-driven interaction. It is possible to add additional icons onto this desktop to represent other application programs, documents, files, and resources. The start menu also can be customized to include additional items, such as for launching particular applications, and opening documents or files. Further, a graphic image can be selected as a background of the desktop (called "wallpaper") in place of the default solid color background.

Other aspects of the Windows® 95 shell also are limited in their capacity to present multi-media content enhancements. In

3

particular, the shell provides windowed folder views accessed through the my computer and network neighborhood icons that represent the contents (i.e., files and sub-folders) of a directory or folder in the file system of the computer or a connected local area network (LAN). The folder views have four display modes: large icon, small icon, list and detail. In the icon modes, the folder view displays icons representing the files and sub-folders in a window against a white background. The icons used to represent the files and sub-folders in the folder view are dependent on the type or format of the file, e.g., documents having a Microsoft Word format are represented with an icon indicative of that application program. In the list and detail modes, the folder view displays a text listing of the files and sub-folders in its window also against a white background.

An add-on product for the Microsoft Windows® 95 operating system, called the Microsoft Windows® 95 Plus Pack, includes packaged enhancements called "themes" for the graphical user interface of the Windows® 95 operating system. Each theme includes a group of resources which alter the appearance and feel of the graphical user interface. These resources include substitute icons, mouse pointer graphics and animations, sounds, a wallpaper, and a screen saver. A particular theme can be selected and applied to the graphical user interface using a themes applet which the plus pack installs into the Windows® 95 control panel (an application program group which includes small application programs or applets that control various aspects of the operating system). Again however, the themes provide only limited multi-media content enhancements to the desktop. The applied theme can change the desktop's wallpaper, and the graphics of the my computer, network neighborhood and recycle bin icons.

In contrast to the limited capacity for multi-media enhancement on the desktop, multi-media content commonly appears in a windowed application program in the graphical user interfaces of Windows® 95 and like operating system shells. Application programs which present multi-media content include desktop publishing, video games, multi-media encyclopedias and like references, Internet browsers, and many others. Since the windowed application programs are separate and independent of the desktop, the multi-media content presented in the application program windows cannot effectively enhance the presentation on the desktop itself.

Further, multi-media content is made available in numerous formats. Still images are available in JPEG (Joint Photographic Experts Group), GIF, BMP (Windows® bitmap), and other file formats. Sounds are distributed in WAV (wave), MIDI and other file formats. Video is distributed in MPEG (Motion Picture Experts Group), AVI and other file formats. The hypertext markup language (HTML) format is widely used to distribute documents or pages including text, images, video and sound on the World-Wide Web of the Internet. Three dimensional environments are now being developed in virtual reality markup language (VRML) and other formats. These various multi-media formats provide a facility for expressing multi-media content, but do not of themselves provide a facility for providing desktop functionality with multi-media enhancement.

The present invention provides multi-media content enhancements to the desktop of an operating system's graphical user interface. In one system according to the invention, an operating system shell synthesizes the display for the desktop into a hypertext multimedia document format (the HTML format, for example). The synthesized document includes the graphical icon oriented and menu driven user interface elements of the desktop, and also can include multi-media enhancements, such as text, graphics, sounds, animations, video, hypertext links, etc. These enhancements can add informative or explanatory content to the desktop, or otherwise customize the appearance and/or behavior of the desktop. The shell also acts as a hypertext multimedia document viewing software to display the synthesized document as the desktop in a graphical user interface, preferably as a full-screen background display to a windowing environment.

According to one aspect of the invention, the shell synthesizes the hypertext multimedia document from a template which contains the multi-media enhancements or references to the enhancements. The shell includes a pre-processor which processes the template and produces the synthesized document which is to be displayed as the desktop. This pre-processor converts soft or variable parameters into data in the hypertext multimedia document's format which is output in the synthesized document for display with the multi-media enhancements. The templates also contain document data for output in the synthesized document to cause embedding of a software object or objects in the displayed view of the synthesized document. These software objects implement the functionality of the desktop's graphical icon-oriented and/or ~~and~~ menu driven user interface elements; A1 When the synthesized document is then displayed as the desktop in the graphical user interface, the object(s) provide the user interface features and functionality (e.g., for launching application programs, opening documents and files, drag and drop functionality, etc.) of the desktop. By embedding the objects in the synthesized document, these user interface features are displayed in combination with the multi-media enhancements by the shell.

According to a further aspect of the invention, the shell also synthesizes hypertext multimedia documents for display as the folder views and other displays in the shell's graphical user interface. The hypertext multimedia documents for the various displays are synthesized from templates which are identified in a configuration or ".ini" file. When the user navigates to one of the displays, the shell looks up the appropriate template to use for the display. The shell then processes the template into a hypertext multimedia document with embedded objects to provide the user interface elements required for the display (e.g., the graphical icons and drag and drop functionality in a folder view). The synthesized document is then displayed by the shell. The shell is thus able to provide multi-media content enhancements to these additional shell displays.

Synthesizing the desktop and other displays from templates further allows the multi-media enhancements to be easily and flexibly added and changed. The templates are in the format of the hypertext multimedia documents, and additionally contain directives to replace soft parameters by the pre-processor. The enhancements provided by the templates can be altered by editing the templates using hypertext document editing software or even a text editor. Alternatively, a new template or complete set of templates can be swapped in by changing entries in the configuration file. Accordingly, the overall appearance and behavior of the shell can be immediately changed by substituting the set of templates identified in the configuration files.

In an embodiment of the invention illustrated herein, the synthesized documents for the various displays are displayed in a single designated area or frame in the graphical user interface, such as the full-screen background display area of the windowing environment. As the user navigates from one display (e.g., the desktop) to another (e.g., a folder view), the shell switches the hypertext document displayed in the designated area in a fashion similar to navigating a hyperlink between hypertext documents.

The shell in the illustrated embodiment further operates as viewer or browser of hypertext documents. In addition to the synthesized hypertext documents for shell displays, the shell also displays and navigates between other hypertext documents, such as those available from the Internet. The shell thus extends hyperlink navigation and the rich multi-media content of hypertext documents to the shell's graphical user interface.

Additional features and advantages of the invention will be made apparent from the following detailed description of an illustrated embodiment which proceeds with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of a computer system that may be used to implement a method and apparatus embodying the invention for incorporating multi-media enhancements to an operating system shell.

Fig. 2 is a data flow diagram showing the synthesis of a hypertext multimedia document by an operating system shell for a desktop display in a graphical user interface.

Fig. 3 is a view of a navigation bar incorporated by the shell of Fig. 2 in the folder display of Fig. 7.

Fig. 4 is a block diagram of typical data structures for an object in the computer system of Fig. 1.

Fig. 5 is an object framework in the shell of Fig. 2 which supports presenting a hypertext document incorporating graphical user interface functionality as a desktop display.

Fig. 6 is a view of a desktop display presented by the shell of Fig. 2.

Fig. 7 is a view of a folder display presented by the shell of Fig. 2.

DETAILED DESCRIPTION OF THE ILLUSTRATED EMBODIMENTS

1. Computer Overview

Referring to Fig. 1, an operating environment for an illustrated embodiment of the present invention is a computer system 20 with a computer 22 that comprises at least one high speed processing unit (CPU) 24, in conjunction with a memory system 26, an input device 28, and an output device 30. These elements are interconnected by at least one bus structure 32.

The illustrated CPU 24 is of familiar design and includes an ALU 34 for performing computations, a collection of registers 36 for temporary storage of data and instructions, and a control unit 38 for controlling operation of the system 20. The CPU 24 may be a processor having any of a variety of architectures including Alpha from Digital, MIPS from MIPS Technology, NEC, IDT, Siemens, and others, x86 from Intel and others, including Cyrix, AMD, and Nexgen, and the PowerPc from IBM and Motorola.

The memory system 26 generally includes high-speed main memory 40 in the form of a medium such as random access memory (RAM) and read only memory (ROM) semiconductor devices, and

8

secondary storage 42 in the form of long term storage mediums such as floppy disks, hard disks, tape, CD-ROM, flash memory, etc. and other devices that store data using electrical, magnetic, optical or other recording media. The main memory 40 also can include video display memory for displaying images through a display device. Those skilled in the art will recognize that the memory 26 can comprise a variety of alternative components having a variety of storage capacities.

The input and output devices 28, 30 also are familiar. The input device 28 can comprise a keyboard, a mouse, a physical transducer (e.g., a microphone), etc. The output device 30 can comprise a display, a printer, a transducer (e.g., a speaker), etc. Some devices, such as a network interface or a modem, can be used as input and/or output devices.

As is familiar to those skilled in the art, the computer system 20 further includes an operating system and at least one application program. The operating system is the set of software which controls the computer system's operation and the allocation of resources. The application program is the set of software that performs a task desired by the user, using computer resources made available through the operating system. Both are resident in the illustrated memory system 26.

In accordance with the practices of persons skilled in the art of computer programming, the present invention is described below with reference to acts and symbolic representations of operations that are performed by computer system 20, unless indicated otherwise. Such acts and operations are sometimes referred to as being computer-executed. It will be appreciated that the acts and symbolically represented operations include the manipulation by the CPU 24 of electrical signals representing data bits which causes a resulting transformation or reduction of the electrical signal representation, and the maintenance of data bits at memory locations in memory system 26 to thereby reconfigure or otherwise alter the computer system's operation, as well as other processing of signals. The memory locations where data bits are maintained are physical locations that have particular electrical, magnetic, or optical properties corresponding to the data bits.

2. Shell Overview

With reference to Fig. 2, a shell 50 for an operating system of the computer 20 (Fig. 1) provides a graphical user interface for a user of the computer to interact with the operating system. The graphical user interface includes a desktop display 52 presented on a video screen of the computer's output device 30 (Fig. 1). The desktop display 52 preferably is presented by the shell in a windowing environment as a full-screen background display. Specifically, any visible windows that are not minimized to an icon are displayed overlaying the desktop display 52 in the windowing environment.

In the illustrated shell 50, the shell presents a variety of views 54 (listed in table 1 below) for different aspects of the graphical user interface on the desktop display 52. These views 54 are similar to displays presented in the graphical user interface of the Microsoft Windows® 95 operating system, but additionally include multi-media enhancements incorporated in the views in accordance with the invention. More particularly, the views 54 include the desktop and desktop container displays of the Microsoft Windows® 95 operating system. The desktop container displays are sub-displays of the Microsoft Window® 95 desktop, that contain a group of related graphical icons for file management or for launching application program and operating system services (e.g., My Computer, Network Neighborhood, Control Panel, etc.).

For each of the views 54 presented on the desktop display 52, the shell 50 synthesizes the view as a hypertext page 56. The illustrated hypertext page 56 is in a hypertext markup language (HTML) format. The HTML format is a standard format for multi-media hypertext documents which is used on the Word-Wide Web portion of the Internet. (The Internet is a global network of cooperatively interconnected computer networks, consisting of millions of individual computers. A portion of the Internet referred to as the "World-Wide Web" consists of computers, also referred to as "sites," which make multi-media

10

documents in HTML format generally available for downloading or retrieval by individuals having a computer with Internet access.) HTML format documents are ASCII encoded files which employ tags to designate text formatting, hyperlinks, and multi-media content to be incorporated from other resources (e.g., files) into the document. Further details of the HTML format of the illustrated hypertext page 56 are described in the <u>HTML Reference</u>, attached hereto as Appendix Q.

A hypertext document is a document that contains hyperlinks. Hyperlinks are references to other documents which are generally indicated in a displayed view of the document as a graphic, underlined text, or highlighted text, and which can be activated by user input to the viewing software to cause the viewing software to navigate to the referenced document. A multimedia document is a document which contains or incorporates multiple forms and/or formats of information content such as text, images, sounds, software objects, video, animations, etc.

The shell 50 obtains the hypertext page 56 from which a view in the graphical user interface is synthesized from processing a stored hypertext template, or alternatively directly from a stored hypertext page. In the former case, the shell 50 includes a pre-processor 60 which synthesizes the hypertext page 56 from one or more of a set of templates 62 and one or more desktop interface controls 64. The templates 62 are files which contain data in the HTML format which is to be incorporated into the hypertext page 56, and additionally include pre-processor directives. The directives are instructions to the pre-processor for converting soft parameters into html formatted data in the hypertext page 56. The templates for each of the views in the illustrated shell are shown in the following Table 1.

T11CX

Table 1. Hypertext View Templates

Friendly Name	File Name
Briefcase	brfcase.htm
Control Panel	control.htm
Default folder	folder.htm

11

Desktop	desktop.htm
Favorites folder	favorite.htm
File-system folder	directry.htm
My Computer	mycomp.htm
Network Neighborhood	nethood.htm
Printers	printer.htm
Recycle Bin	recycle.htm
Start Menu and subfolders	startmnu.htm
Workgroups	domain.htm
Vendor-specific workgroups	msdomain.htm
Vendor-specific networks	msnet.htm
Vendor-specific servers	msserver.htm
Servers	server.htm
Audio CD	audiocd.htm
Dial-Up Networking	dialupnt.htm
Entire Network	fullnet.htm
Fonts folder	fonts.htm
History	history.htm
My Documents	mydocs.htm
Network Workgroup	workgrp.htm
Program Files folder	progfile.htm
Root of data CD	datacd.htm
Root of floppy disk drive	floppy.htm
Root of hard disk	harddrv.htm
Windows folder	windows.htm

In general, the templates listed in Table 1 are related one-to-one with folders and are used to synthesize a hypertext page for a display associated with the related folders. Some of the folders correspond to actual directories in a file system of the computer's memory system 26 (Fig. 1). For example, each of the "windows folder," "root of hard disk," and "my documents" folders correspond to actual file system directories. The displays associated with these folder generally represent (at least in part) the contents of the corresponding directory, and are called "folder views." Others of the folders (termed "virtual folders") do not correspond to any file system directory. Accordingly, the displays associated with these folders generally do not represent the contents of a file system directory. For example, the "my computer" folder is a virtual folder.

The templates listed in Table 1 are for producing displays associated with a set of standard folders in the Windows® operating system. In addition to these standard folder templates listed in table 1, the templates 62 also can include additional templates for non-standard folders (herein called "custom templates"), such as folders corresponding to file system directories created by a user or added by a software installation program. For example, an installation program of an application software product (such as a productivity software, computer game, or utility software) that creates a new folder in which to install the application software's files also can add a template associated with the folder to the set of templates 62. When the newly created folder is viewed in the graphical user interface, the shell 52 uses this added custom template to produce a folder view display representing the newly created folder's contents. These added custom templates can contain multimedia content enhancements specific to the new folder, such as graphic images, text, hyperlinks, or software objects relating to the application software product or its vendor.

The template (or alternatively stored hypertext page) to be used by the shell in synthesizing the hypertext page view in each display in the graphical user interface is identified in one or more configuration files 66. The configuration files 66 can include both local and global configuration files. More particularly, folders that are actual file system directories can contain a hidden local configuration file (named "desktop.ini" in the illustrated computer 20). (Hidden files are files having a flag or attribute which is set to indicate that the file normally is not displayed by a file management tool, e.g., the Windows Explorer in the Windows® 95 operating system.) This "desktop.ini" configuration files stores data identifying the template (e.g., by path and file name in the computer's file system) to be used in producing a folder view display

13

of the folder. A listing of a representative desktop.ini file is shown in the following table 2.

T14jcX

Table 2. Representative Desktop.Ini File Listing.

```
[ExtShellFolderViews]
Default={FB7E5040-1F6D-11D0-89A9-00A0C9054129}
{FB7E5040-1F6D-11D0-89A9-00A0C9054129}={FB7E5040-1F6D-11D0-
89A9-00A0C9054129
}
{00000001-0001-0002-0003-000000000001}={25336920-03F9-11CF-
8FD0-00AA00686F13
}
{00000002-0001-0002-0003-000000000002}={00020900-0000-0000-
C000-000000000046
}
{00000003-0001-0002-0003-000000000003}={00020810-0000-0000-
C000-000000000046
}

[{00000001-0001-0002-0003-000000000001}]
PersistFile="pageone.html"
MenuName="friendly-name-for-view-1"
ToolTipText="Html View"
HelpText="This shows a HTML document"

[{00000002-0001-0002-0003-000000000002}]
PersistFile="word.doc"
MenuName="friendly-name-for-view-2"
ToolTipText="Word Document View"
HelpText="This shows a Word document"

[{00000003-0001-0002-0003-000000000003}]
PersistFile="Excel.xls"
MenuName="friendly-name-for-view-N"
ToolTipText="Excel Spreadsheet view"
HelpText="This shows an Excel spreadsheet"

[{FB7E5040-1F6D-11D0-89A9-00A0C9054129}]
IconArea_Image="c:\win95B\bubbles.bmp"
IconArea_Pos=1
```

The above representative desktop.ini file begins with a section having the heading "[ExtShellFolderViews]." This section lists globally unique identifiers ("GUIDs") associated with software objects that implement views of the folder in which the desktop.ini file is stored. A line beginning "default=" specifies a default view of the folder. The

14

four lines below this specify alternative custom views of the folder in the format, <GUID> = <GUID>. The left GUID on each line identifies the software object that implements the view. If there is an entry with this left GUID in the system registry for the view object (i.e., the GUID is a CLSID registered in the system registry), then the line modifies some of the view's attributes. In such case, the right GUID is a CLSID that identifies the pre-processor 60 to be used for processing the template for that view, or alternatively identifies the viewer object (e.g., the HTML viewer 70 of Fig. 2 or other document object) which displays the hypertext page in the view (i.e., for use when the hypertext page itself is stored rather than a template from which the hypertext page is produced). If there is no entry with the left GUID in the system registry, then the line indicates a custom view and is unique only with the respective desktop.ini file.

More specifically, in the above representative desktop.ini file, the first or default line of the "[ExtShellFolderViews]" section identifies an object that implements a default view for the folder. The second line overrides some attributes of one of the folder's views. A separate section at the bottom of the desktop.ini file has lines which change the "IconArea_Image" and the "IconArea_Pos" attributes of that view, specifically the background bitmap of the view and its position (e.g., whether centered or tiled).

The third, fourth and fifth lines of the "[ExtShellFolderViews]" section specify custom views for the folder. The left hand identifiers are not CLSIDs of views registered in the system registry, and are unique only within the desktop.ini file. The right hand GUIDs on these lines are CLSIDs of objects registered in the system registry. Specifically, the right hand GUIDs are CLSIDs of the HTML viewer 70, a Microsoft® Word document object, and a Microsoft® Excel document object, respectively. (Document objects are described below.) The desktop.ini file contains a section for each of the custom views which have the line "PersistFile = ..." When one of these views is selected for display, the respective document object of the view is instantiated with the hypertext page, Word document, or Excel document, respectively, identified on the "PersistFile" line.

The template (or alternatively stored hypertext page) associated with a folder (whether an actual file system directory or virtual folder) also can be identified by entries in a global configuration file, which in the illustrated shell 52 is the system registry. In the Microsoft® Windows® operating system, the system registry is a database which stores configuration information for the operating system, including information to enumerate and track applications, device drivers, and operating system control parameters. For a detailed discussion of the registry, see *Win32 Programmers Reference, Vol. 2*, published by Microsoft Press, Redmond, Washington (1993). Representative entries in the system registry for the illustrated shell 52 are listed in the following Table 3.

T16X

Table 3. Representative System Registry Entries for Folder Views

Folder Type	Registry Entry
Virtual Folder	HKCR\CLSID\{guid of virtual folder}\shellex\ExtShellFolderViews\{guid of view} PersistFile="template.htm"

Directory	HKCR\Directory\shellex\ExtShellFolderViews\{guid of view}
	PersistFile="directry.htm"

Default Folder	HKCR\Folder\shellex\ExtShellFolderViews\{guid of view}
	PersistFile="folder.htm"

Each of the representative registry entries in the above table include a GUID of a view which identifies either a pre-processor (where the view is synthesized from processing a template) or a document viewer (where the view is synthesized directly from a stored hypertext page or other document). If the GUID identifies a pre-processor (e.g., the pre-processor 60) then the registry also contains an entry to identify the document viewer (e.g., the hypertext viewer 70 of Fig. 2) for the hypertext page or other document produced from the pre-processor processing the template. This registry entry for the pre-processor has the form:

```
HKCR\CLSID\{guid of view}\FilterView = "{guid of document
viewer}"
```

These system registry entries that identify templates (or stored hypertext page or other document) to use in producing displays of the graphical user interface are created when the operating system shell 50 is first set up or installed. The registry entries can be edited manually or altered by software to substitute a different set of templates for the standard templates listed in Table 1 or to add to the standard templates.

The shell 50 determines the template (or stored hypertext page) to use in synthesizing the view of a particular folder from this information in the configuration files 66. If the folder is an actual file system directory, the shell 50 checks for a desktop.ini configuration file in the directory. If the folder contains a desktop.ini configuration file which identifies a template (or stored hypertext page or other document), the shell 50 uses that template. If not, the shell 50 checks in the system registry for a registry entry under the "HKCR\directory" key (as shown in the representative registry entry for a directory in Table 3 above) that is specific to the folder and identifies its associated template (or stored document). If such a folder specific registry entry is found, the shell 50 uses the template identified in that folder-specific entry. If not, the shell 50 checks the system registry for a default folder entry under the "HKCR\folder" key (as shown in the representative default folder registry entry in Table 3 above). If the default folder entry is found, the shell 50 uses the template or document (e.g., the "folder.htm" template listed in the above Table 1) identified in the default folder entry. This default folder registry entry can be edited to change the default folder template (or document) used when no folder specific template is found.

If, on the other hand, the folder is a virtual folder, the shell 50 checks for an entry under the HKCR\CLSID key that lists a class identifier of the virtual folder (as shown in the representative registry entry for a virtual folder in Table 3 above). If such a virtual folder entry is found that identifies a template (or document), the shell 50 uses the identified template to produce the virtual folder's hypertext page 56. Otherwise, the shell 50 looks to the default folder entry under the "HKCR\folder" key (as shown in Table 3 above), and uses the default folder template identified in that entry.

The particular folder for which the shell 50 synthesizes the current view 54 in the desktop display 52 is controlled by user action. At start-up, the shell synthesizes the desktop view from the "desktop.htm" template for the desktop folder. From this view, the user can navigate to other views (e.g., folder views of the my computer, network neighborhood, or file system directory folders) by activating icons or hyperlinks on the desktop display, or selecting commands from menus available on the current page. For example, the user can navigate to the my computer view by mouse clicking on the My Computer icon on the desktop view. Whereupon the shell 50 synthesizes the My Computer view from the "mycomp.htm" template, and displays the My Computer view.

With reference to Fig. 3, a navigation toolbar 200 is presented by the shell 51 on folder view displays, such as the My Computer view. The user activates controls on the navigation bar to move between folder view displays, or back to the desktop display. The navigation toolbar 200 includes a drop down list control 202 in which the user can select to navigate to a desired folder. The destination folder can be selected from a list or typed into the control 202. The navigation toolbar 200 also includes back and forward history controls 204 which the user can activate to move backwards and forwards through a history of previously navigated folders. The navigation toolbar 200 further includes a favorites menu button 206 which accesses a user specified list of folders. Each folder view display also may include other controls for navigating to another folder view display. The My Computer view, for example, contains icons which the user can activate to navigate to the floppy and hard disk drive folder views.

When synthesizing the hypertext page 56 for the current view, the pre-processor 60 parses the corresponding template(s) for the view (as shown in the Table 1 above or identified by the "desktop.ini" configuration file), and performs any pre-processor directives encountered in the templates. In the illustrated templates 62, the pre-processor directives follow a syntax rule of being bracketed between the characters "<%" and "%>" to distinguish from HTML formatted data. The directives primarily specify soft parameters which the pre-processor converts into HTML formatted data for output into the hypertext page 56. For example, the soft parameters can include the name of the folder whose contents are to be shown in a folder view, the name of the My Computer or other view (the user can rename these icons and corresponding views in the illustrated shell). Further details of the pre-processor directives are described below.

The pre-processor 60 utilized by the shell 50 for processing the templates 62 into hypertext pages 56 for use as displays in the graphical user inteface is identified by entries in the system registry. A class identifier ("CLSID") associated with the pre-processor is stored in the system registry under an "ExtShellFolderViews" key, and given a named value of "PersistFile = desktop.htm." The pre-processor 60 is implemented as an OLE/COM object (as described below). The pre-processor also has an associated entry under its CLSID in the system registry. This entry has a subkey "FilterView" under which is stored a CLSID associated with the document viewer (e.g., the HTML viewer 70 in the illustrated shell 50). This allows a third party software vendor to substitute a custom pre-processor into the shell 50 in place of the illustrated pre-processor 60.

The desktop interface controls 64 are software components (referred to as "objects") that implement the functionality of a particular aspect of the shell's graphical user interface. For example, the desktop interface control for the desktop view provides the graphical icons (e.g., My Computer, Network Neighborhood, and Recycle Bin icons), menus, and functionality (e.g., drag and drop, icon and menu activation, and other operations) of that view. The control which is incorporated into the hypertext page 56 for the folder views provides the large icon, small icon, list and detail mode graphical icon oriented user interface and functionality of the folder views in the Microsoft Windows® 95 operating system. These controls include interfaces which allow the controls to interoperate with the shell to provide the view on the desktop display 52, and with the operating system to provide file management and to launch application programs and operating system services. The desktop interface ~~controls 60~~ controls 64 are embedded in the hypertext page 56 using HTML tags as described in further detail below.

The illustrated shell 50 also includes a hypertext viewer 70 and a shell explorer 72 which provide the desktop display 52 on the video screen of the computer's output device 30 (Fig. 1). The hypertext viewer 70 is a software component in the shell 50 which operates to parse the HTML formatted data in the hypertext page 56, and produce the view 54 of the hypertext page for display in the desktop display 52. In effect, the hypertext viewer 70 contains the HTML parsing and display code equivalent to an Internet browser. The illustrated hypertext viewer supports parsing and display of the same HTML format which is supported in the Microsoft® Internet Explorer.

The shell explorer 72 is another software component of the shell which manages graphical user interface elements of the desktop display other than the view. More specifically, the shell explorer 72 provides a frame of the desktop display having a display area in which the hypertext viewer 70 draws the view 54 of the hypertext page 56. The frame includes user interface elements (e.g., the task bar 78 with start button and menu) that continue to be displayed on the desktop display when the hypertext view changes. In effect, the shell explorer 72 acts as a host or container of the hypertext page view 54 displayed by the hypertext viewer 70. The shell explorer 72 and hypertext viewer 70 cooperate to embed the view 54 provided by the hypertext viewer 70 in the desktop display's frame provided by the shell explorer 72.

3. Object Overview

With reference to Fig. 3, the shell explorer 72 and the hypertext viewer 70 are implemented as objects which conform to Microsoft Corporation's Component Object Model (COM), and support various ActiveX (also known as Object Linking and Embedding ("OLE"))

2C

interfaces. COM, ActiveX and OLE are object-oriented technologies which provide integration and interoperability between separate software components. For a detailed discussion of OLE see Inside OLE, Second Edition by Kraig Brockschmidt, Microsoft Press, Redmond, Washington 1995. A brief overview of objects in OLE and associated terminology is provided below before discussing the details of the illustrated embodiment.

Using these object-oriented technologies, the illustrated shell 50 (Fig. 2) works with the hypertext page 56 by encapsulating the hypertext page into an associated object (i.e., the hypertext viewer 70), and integrating with the object using pre-defined interfaces (hereafter referred to as the document object interfaces, and also referred to as the IOLEXXX/IMSOXXX interfaces, which are described in more detail in the OLE Document Objects Specifications, attached hereto as Appendix R). The hypertext viewer document object includes code to work with the hypertext page 56 (i.e., data in the HTML format), including code to display a view of the hypertext page in the desktop display 52. The document object interfaces of the hypertext viewer allow integration with the shell explorer 72, which operates as a host or container of document objects (i.e., by providing the desktop display 52 having a display area in which a hosted document object - the hypertext viewer 70 - can display its view of the hypertext page 56). As a document object host, the shell explorer 72, can host any variety of document (e.g., documents associated with the Microsoft Word, Microsoft Excel, and other application programs from Microsoft or other software developers) which is encapsulated by an object that supports the document object integration interfaces. Implementing the hypertext viewer as a document object also allows the desktop and folder views of the synthesized hypertext page 45 to be embedded in other document object containers, such as a file manager, Internet browser or other application program which is implemented as a document object container. An example in which a view of the hypertext page 56 produced by the hypertext viewer 70 is hosted in the desktop display 150 of the shell explorer 72

21

is shown in Fig. 5 and described in more detail below. A further example in which a view of the hypertext page 56 produced by the hypertext viewer 70 is hosted in a window of an application is shown in Fig. 6 and described in more detail below.

An object is an instance of a programmer-defined type referred to as a class, which exhibits the characteristics of data encapsulation, polymorphism and inheritance. Data encapsulation refers to the combining of data (also referred to as properties of an object) with methods that operate on the data (also referred to as member functions of an object) into a unitary software component (i.e., the object), such that the object hides its internal composition, structure and operation and exposes its functionality to client programs that utilize the object only through one or more interfaces. An interface of the object is a group of semantically related member functions of the object. In other words, the client programs do not access the object's data directly, but must instead call functions on the object's interfaces to operate on the data.

Polymorphism refers to the ability to view (i.e., interact with) two similar objects through a common interface, thereby eliminating the need to differentiate between two objects. Inheritance refers to the derivation of different classes of objects from a base class, where the derived classes inherit the properties and characteristics of the base class (which for purposes of OLE are the interfaces of the base class).

Microsoft Corporations's COM specification defines binary standards for objects and their interfaces which facilitate the integration of software components. According to the COM specification, a typical object 80 is represented in the computer system 20 (Fig. 1) by an instance data structure 82, a virtual function table 84, and member functions 86-88. The instance data structure 82 contains a pointer 90 to the virtual function table 84 and data 92 (also referred to as data members, or properties of the object). A pointer is a data value that holds the address of an item in memory. The virtual function table 84 contains entries 96-98 for the member functions 86-88. Each of the entries 96-98 contains a reference to the code 86-88 that implements the corresponding member function.

The pointer 90, the virtual function table 84, and the member functions 86-88 implement an interface of the object 80. Client programs interact with the object 80 by obtaining a pointer (referred to as an interface pointer) to the pointer 90 of the virtual function table 84. OLE includes a type definition of an interface pointer which allows client programs to call member functions on the interface by name through the interface pointer and provides type checking on the function's arguments, as expressed in the following code (in the C++ programming language):

```
pInterface->MemberFunction(...)
```

By convention, the interfaces of an object are illustrated graphically as a plug-in jack as shown for the document object in Fig. 3. Also, Interfaces conventionally are given names beginning with a capital "I." Objects can include multiple interfaces which are implemented with one or more virtual function tables. The member function of an interface is denoted as "IInterfaceName::FunctionName."

The object 80 conforming to the COM specification exhibits data encapsulation by exposing its interfaces (semantic groupings of its member functions) to client programs. The client programs interact with the object 80 by calling the member functions 86-88 on a particular interface of the object, but do not directly manipulate the object's data. The object 80 also exhibits polymorphism and inheritance in that the object 80 can provide interfaces in common with a base class and other similar objects, so that client programs can interact with each of the objects in the same manner by calling member functions of the interface that the objects have in common.

4. Document Object and Server Overview

Referring still to Fig. 3, the virtual function table 84 and member functions 86-88 of the object 80 are provided by a server application program 100 which is stored in the computer system 20 (Fig.

1) as an executable program file (with a ".exe" file name extension) or as a dynamic link library file (with a ".dll" file name extension). Dynamic link library files are loaded, dynamically linked, and executed by the Windows® operating system in a same process with a client application program. Executable program files are loaded by the operating system as a separately executing process. In accordance with OLE, the server application 100 includes code for the virtual function table 84 (Fig. 3) and member functions 86-88 (Fig. 3) of the classes that it supports, and also includes a class factory 102 that generates the instance data structure 82 (Fig. 3) for an object of the class.

A server application can be written by a programmer to support a particular class of object that contains any desired data. More specifically, a programmer can write server applications which provide objects that contain the data of a particular variety of computer document (e.g., document 56 ~~document 60~~ of Fig. 2), such as a text document, spreadsheet, drawing, etc., or that contain data for part of a computer document, such as a range of spreadsheet cells, a paragraph of a text document, etc. These objects which contain document data (and additionally support the document object interfaces described above) are referred to herein as document objects. (Further details of document object are described in the OLE Document Objects Specifications, attached hereto as Appendix R.) For example, software application programs such as Microsoft® Word can be written as a server application in which the application program's documents are represented as OLE objects. The illustrated shell 50 (Fig. 2) includes a server application which represents the hypertext page 56 as a document object (i.e., the hypertext viewer 70). This allows the shell explorer 72 (Fig. 2) and other document object containers to interact with the hypertext page 56 through the document object interfaces.

For a client program (e.g., a document object host or container) to interact with the object 80 provided by the server application 100, the server application must first create the object (i.e., instantiate an object of a class supported by the server application) and the client program must gain an interface pointer to the object 80. In

OLE, the client program realizes these events using services provided by OLE and a set of standard object interfaces defined by COM based on class and interface identifiers assigned to the object's class and interfaces. More specifically, the services are available to client programs as application programming interface (API) functions provided in the COM library, which is part of a component of the Windows® operating system in a file named "OLE32.DLL." In OLE, classes of objects are uniquely associated with class identifiers ("CLSIDs"). Class identifiers are 128-bit globally unique identifiers ("GUID") that the programmer creates with an OLE service named "CoCreateGUID" and assigns to the respective classes. The interfaces of an object are associated with interface identifiers ("IIDs").

In particular, the COM library provides an API function, "CoCreateInstance," that the client program can call to request creation of an object to encapsulate a particular documents data using a CLSID associated with the data. The CoCreateInstance API function creates the object and returns a pointer of the requested interface to the client program.

Once the client program has obtained a first interface pointer to the object 80, the client program obtains pointers to other desired interfaces of the object using the interface identifier associated with the desired interface. COM defines several standard interfaces generally supported by OLE objects including the IUnknown interface. This interface includes a member function named "QueryInterface." The QueryInterface function can be called with an interface identifier as an argument, and returns a pointer to the interface associated with that interface identifier. By convention, the IUnknown interface's member functions are included as part of each interface on an object. Thus, any interface pointer that the client program obtains to an interface of the object 80 can be used to call the QueryInterface function.

In a typical situation, however, the only information that the client program has to reference a particular document is a textual name, such as a file name or an Internet URL. In the case of a file name, the COM library provides API functions (e.g., "GetClassFile,"

25

"ReadClassStg" and "ReadClassStm") for obtaining a CLSID associated with the file. Client programs also can utilize a system provided object known as a moniker to resolve a name that references a document into an interface pointer on an instantiated object that encapsulates the document's data. These well known mechanisms are described in more detail in Inside OLE, Second Edition, supra.

5. Shell Objects and Interfaces

In addition to the document object interfaces, the illustrated shell 50 (Fig. 2) also utilizes an object-oriented framework of the Windows Explorer application, folder view windows, and "File Open" dialog in the Microsoft Windows® 95 operating system. This framework defines three interfaces, a shell browser ("IShellBrowser") interface 122, a shell view ("IShellView") interface 126 and a shell folder ("IShellFolder") interface 128, which are described in further detail in the Appendix S attached hereto. The shell browser interface 122 is supported on an object which acts as a container for views of a namespace, such as the folders in a file system. This container object provides a top level window having a frame in which to embed the namespace view. Examples of such a container object include the object which provides the top level or outer window in the Windows Explorer application, and also the object which provides the top level window for folder views in the shell of the Windows® 95 operating system. The namespace is a collection of symbols (e.g., names) associated with items managed through the container object, and rules for resolving a given name into the item it designates. In the case of an operating system shell, the namespace may include file names, directory names, storage devices, registry keys, printers, network resources, etc.

The shell browser interface 122 allows the namespace view contained in such a container window to insert menu items with those of the container window into a composite menu, install the composite menu into the container's window, and also remove the container's menu items from the composite menu. The shell browser interface 122 also allows the namespace view to set and display status text relevant to the view in the container's window. The view also can enable and disable the container's modeless dialog boxes, and translate accelerator keystrokes intended for the container through functions of the container object exposed by the shell browser interface 122.

The shell view interface 126 is supported on a view object which provides a namespace view for display in the container object's window. In the Windows® 95 operating system, for example, includes a default view ("DefView") object 124 which provides a view of the desktop in the Windows® 95 shell's graphical user interface which graphically represents the desktop, including the graphical icon-oriented and menu user interface elements of the desktop view (i.e., the full screen background display with the My Computer, Network Neighborhood, and Recycle Bin icons, drag and drop functionality, and pop-up menus of the desktop). Other view objects in the Windows® 95 operating system provide the large icon, small icon, list and details views representing a folder's contents which are displayed in the folder view windows and the Windows Explorer application's window. The container object for these windows communicate with the view objects through the shell view interface 126. The communication involves the translation of window messages, the state of the container object's window (e.g., activated or deactivated), the merging of menus, and tool bar items.

The shell folder interface 128 is provided in this framework for implementing on objects that manage or extend the namespace. The shell folder interface 128 provides functions to display or operate on the contents of the namespace. For example, an object supporting the shell folder interface can add additional groups of symbols to the namespace, such as uniform resource locators ("URLs") of World-Wide Web pages.

In the illustrated shell 50, the shell browser interface 122 is implemented on a shell explorer object 120 which provides the desktop display 52 (Fig. 2) and acts as a container for the hypertext view 54. As described above, the desktop display 52 operates as the desktop of a windowing environment of a graphical user interface provided by the shell 50. More particularly, the shell explorer object 120 in the illustrated shell 50 is the Shell Explorer from the Microsoft Windows® 95 operating system which manages the Windows® 95 operating system's

27

overall graphical user interface and windowing environment, and acts as a container for the desktop and folder views provided by the default view object 124.

The illustrated shell 50 (Fig. 2) extends the framework of the Windows® 95 operating system shell to also support document object hosting by the shell explorer 120, so as to allow displaying the view 54 of the synthesized hypertext page 56 in the desktop display 52. This is achieved in the illustrated shell 50 by a web view object 130 which supports the document object container interfaces 132 in addition to the shell view interface 126 and the shell folder interface 128. By supporting these interfaces, the web view object 130 effectively operates as a bi-directional proxy object. The web view object 130 can act as both a shell view object that can by hosted by the shell explorer 120, and as a document object container that can host the hypertext viewer 70 or other document objects. Since the web view object 130 supports the shell view and shell folder interfaces, it can take the place of the default view object 124 in the framework of the Windows® 95 operating system shell.

6. Desktop Interface Controls

With reference still to Fig. 4, the default view object 124 (Fig. 2) from the Windows® 95 shell framework is modified in the illustrated shell 50 for embedding in the hypertext page 56 as one of the desktop interface ~~controls 60~~ controls 64. More specifically, the desktop interface controls are implemented as ActiveX controls. An ActiveX control is an object which conforms to the COM specification, and may support additional OLE interfaces such as those for drag and drop operations and other OLE functions. In particular, the desktop interface ~~controls 60~~ controls 64 (including the default view object 124) support OLE interfaces that allow the object to be embedded in a document object. In other words, interfaces which allow a document object such as the hypertext viewer 70 to act as a container or host of the embedded desktop interface ~~controls 60~~ controls 64. Further details of ActiveX controls are described in in the OLE Control and Control Container Guidelines, attached hereto as Appendix T.

In the illustrated shell 50 (Fig. 2), the desktop interface controls 60 are embedded in the hypertext page 56 using an HTML object embedding tag, which has the following format:

```
<OBJECT classid= codebase= data= height= width=>
```

The classid parameter of this tag (if present) specifies a class identifier of the control. The hypertext viewer uses the class identifier to create the control, such as by calling the CoCreateInstance API function to cause the control's server application to be loaded and the server application's class factory to instantiate the control. The codebase parameter (if present) specifies a URL of a file containing the control (such as on the Internet). If the control is not installed on the computer 20, the hypertext viewer can retrieve this file using the URL from the Internet and then install the file on the computer before instantiating the control using its class identifier. The data tag (if present) specifies persistent data for the control as either a text string or via a URL of a file containing the control's persistent data. Further details of the HTML object embedding tag are described in the <u>HTML Reference</u>, attached hereto as Appendix Q.

The templates 62 which are processed into the hypertext page 56 include the HTML object embedding tags which specify the desktop interface ~~controls 60~~ controls 64 to embed in the displayed hypertext view 54. Since the HTML object embedding tags already are in the HTML format, the pre-processor 60 outputs the tags along with other HTML format data from the templates 62 directly into the hypertext page 56 without conversion.

When displaying the hypertext view 54 of the hypertext page 56, the hypertext viewer 70 parses the HTML object embedding tags along with the other HTML format data in the hypertext documents. On encountering the HTML object embedding tags for the desktop interface ~~controls 60~~ controls 64 during the parsing, the hypertext viewer 70 instantiates the desktop ~~controls 60~~ controls 64 using the class identifiers specified in the tags. If the server applications for the ~~controls 60~~ controls 64 are not installed on the

29

computer 20 (Fig. 1), the hypertext viewer 70 can download the server application using the URLs specified as the codebase attribute of the tags (if any). The hypertext viewer 70 then displays the instantiated desktop interface controls 64 ~~controls 60~~ together with the other content of the hypertext page in the hypertext view 54. In addition to the desktop interface controls 64 ~~controls 60~~ which implement the desktop and folder view functionality of the shell, other ActiveX controls also can be incorporated in the same manner to provide further multi-media enhancements in the form of executable software content.

a

a

7. Template Pre-Processor Directives

As described above with reference to Fig. 2, the templates 62 contain directives which control processing of the templates by the pre-processor 60 into the hypertext page 56. The directives can cause the pre-processor to perform several different operations, which include converting parameters with variable values into HTML format data for output into the hypertext page 56, processing additional templates into sub-pages of the hypertext page 56, and also specifying a particular pre-processor to use for the pre-processing. As described above, the directives each begin with the characters "<%" and end with the characters "%>" to distinguish from the HTML data in the templates 62.

Each of the illustrated templates 62 begin with the following initial directive, which the pre-processor 60 uses to detect whether the template is a valid template and to detect its beginning.

```
<%WinShell = CreateObject("WinShellHTMLPreProc")%>
```

As this initial directive indicates the beginning of the template, the pre-processor 60 processes only directives that appear after this initial directive in the template and ignores any directive appearing on any lines before the initial directive in the template. If this initial directive is not encountered in the template, the template is considered invalid.

The later directives in each template specify parameters whose value the pre-processor converts into an HTML format text string for output into the hypertext page 56. In the illustrated templates 62,

30

these directives have the format of the text string "WinShell." followed by a tag which designates the particular parameter to be converted, as shown in the following example directives:

T312X

```
<%WinShell.Title%>
<%WinShell.TemplateDirPath%>
<%WinShell.ProcessFile("dsk_1.htm")%>
<%WinShell.IfProxy("http://iptdweb.microsoft.com/activedeskt
op/desktop/tickhost.htm","%WINDIR%\web\ticker.htm")%>
```

For some of these directives, the conversion of parameters into HTML format text strings by the pre-processor 60 is as simple as outputting a current value of the parameter in the form of a text string. For example, the illustrated pre-processor 60 responds to the "<%WinShell.Title%>" and the "<%WinShell.TemplateDirPath%>" directives by simply outputting a text string of the designated parameter's current value. The tag "Title" refers to a parameter whose value is a name of the view which is produced from the template, e.g., "Desktop" for the desktop view, or the name of a particular folder for a folder view. The tag "TemplateDirPath" refers to a directory path name of the directory in the file system where the templates 62 are stored.

Other directives require move complex processing by the pre-processor 60 for conversion to an HTML format text string. The "<%WinShell.ProcessFile("dsk_1.htm")%>" directive for example, causes the pre-processor 60 to also process another of the templates 62 (e.g., the "dsk_1.htm" file specified in the tag) into an HTML format file, then output the name of that file in the form of an HTML text string as the value of the parameter. As another example, the directive with the tag "IfProxy" instructs the pre-processor to output a first text string if the shell 50 is running on a computer which uses a proxy server on a local area network for connecting to the Internet, and to output a second text string otherwise.

8. Desktop Hypertext Page View Example

Fig. 5 illustrates an example hypertext desktop view 150 produced and displayed in the desktop display 52 (Fig. 2) on a video screen of the computer 20 (Fig. 1) according to the illustrated

31

embodiment of the invention. The example hypertext desktop view 150 is produced in the illustrated shell 50 by processing a set of templates using the pre-processor 60 (Fig. 2), including a "Desktop.htm" template, and a "Dsk_1.htm" template which are listed in the attached Appendices A and B, respectively. As a result of this processing, the pre-processor 60 outputs a hypertext page consisting of a "Sfv2395.tmp" and a "Sfv15143.htm" HTML format files which are listed in the attached Appendices G and H, respectively. This synthesized hypertext page additionally incorporates data from the HTML format files, "Infopane.htm," "news.htm," "ticker.htm," and "tickhost.htm" which are listed in the attached Appendices C through F, respectively. The synthesized hypertext page is parsed by the hypertext viewer 70 to generate the example hypertext desktop view 150.

The example hypertext desktop view 150 integrates a variety of multi-media elements with the graphical icon oriented and menu driven user interface elements of the desktop. The example hypertext desktop view 150 has separate HTML frames (i.e., areas on an HTML page) for a ticker 152, a news pane 154, and desktop icons frame 156. The ticker 152 presents scrolling HTML format information retrieved from the Internet, such as stock quotes and sports scores. The news pane 154 also presents HTML format information retrieved from the web, such as images and text relating to news events. The desktop icons frame 156 includes the desktop interface control 64 ~~control 60~~ (Fig. 2) which provides the graphical icon oriented and menu driven user interface elements of the desktop, including the "My Computer," "Network Neighborhood," "Recycle Bin" icons and other user installed icons as well as the drag and drop and other operations of the desktop user interface as in the Windows® 95 operating system shell.

a

9. <u>Folder Hypertext Page View Example</u>

Fig. 6 illustrates an example folder view 170 produced and displayed within an Explorer window 172 on a video screen of the computer 20 (Fig. 1) according to the illustrated embodiment of the invention. The example folder view 170 is produced in the illustrated shell 50 by processing a set of templates using the pre-processor 60

32

(Fig. 2), including a "Directory.htm" template, a "Dir_1.htm" template, a "Dir_2.htm" template, and a "Dir_3.htm" template which are listed in the attached Appendices I through L, respectively. As a result of this processing, the pre-processor 60 outputs HTML format files, "Sfv6190.tmp," "Sfv67916.htm," Sfv47917.htm," and "Sfv47945.htm." The "Sfv6190.tmp" HTML file is a synthesized hypertext page which incorporates HTML format data from the other files. The synthesized hypertext page is parsed by the hypertext viewer 70 to generate the example folder view 150.

The example hypertext folder view 170 is shown hosted in an Explorer application window 172 in Fig. 6, rather than the desktop display 52 (Fig. 2) as was the desktop view 150 (Fig. 5). The Explorer application in this example is implemented using the object-oriented framework shown in Fig. 4. However, in this example, the shell explorer object 72 provides an application window similar to that of the Windows Explorer application in the Windows® 95 operating system. Using the framework, the Explorer application hosts the hypertext viewer 70 as a document object which allows the view 54 of the hypertext page 56 to be displayed in Explorer application's window 172.

The example hypertext folder view 170 integrates a variety of multi-media elements with the graphical icon oriented and menu driven user interface elements of Windows® 95 folder views (i.e., the large icon, small icon, list and detail folder views). The example hypertext folder view 150 represents the contents of the folder, "c:\dos," on the computer 20 (Fig. 1), and has separate HTML frames for a title banner 176, a supplemental hypertext frame 178, and a folder icons frame 180. The title banner 176 presents a title (specifically a name of the folder represented by the folder view) with formatting (e.g., font face, size, etc.) as specified by the HTML data in the hypertext page, and hypertext links which the user can activate to navigate to further hypertext views (i.e., alternative hypertext views of the folder, and a hypertext help page with instructions on operating the Explorer application). The supplemental hypertext frame 178 includes text and hypertext links to supplement the folder's contents. The user can navigate these hypertext links to

33

a

hypertext pages on the Internet. The folder icons frame 180 includes an embedded desktop interface ~~control 60~~ control 64 which provides the graphical icon oriented and menu driven user interface elements of the folder view, including icons representing the files and sub-folders contained in the folder (i.e., the "c:\dos" folder) represented in the folder view as well as the icon drag and drop, scrolling and other user interface operations as are available in the folder views of the Windows® 95 operating system.

The example hypertext desktop view 150 and example hypertext folder view 170 show a few of the multi-media elements that can be incorporated with graphical user interface elements into the hypertext view 54 for display in the desktop display 52 or an Explorer application window. In the illustrated shell 50, the hypertext viewer 70 contains code to parse and display hypertext pages in the HTML format, including any of the HTML tags supported by the Microsoft® Internet Explorer 3.0 browser software. The hypertext views in the illustrated shell therefore can incorporate any of the multi-media elements implemented using these HTML tags, including HTML frames, tables, fonts, hypertext links, embedded images, embedded sounds, embedded objects, and others. Other multi-media elements also can be incorporated in the hypertext views by modifying the hypertext parsing and display code of the hypertext viewer 70 to also support appropriate tags for such elements.

In an alternative embodiment of the invention, the shell 50 can produce displays in the graphical user interface directly from stored hypertext pages. More particularly, in place of a stored set of templates 62 (Fig. 2), the hypertext page for each display is stored and identified in the configuration files 66. This elimates pre-processing of a stored template to produce the hypertext page whenever a display is generated, and thus speeds generation of displays in the graphical user interface. The shell 50 therefore omits the pre-processor 60 in this alternative embodiment.

In some alternative embodiments of the invention, multimedia document formats other than HTML can be used for the pages 56 (Fig. 2). Further, the format of the pages 56 can be other than a hypertext format, or in other words need not allow for hyperlinks to be incorporated into the pages 56. For example, a multimedia document format which does not include hyperlinks, such as the document formats used in the Microsoft® Word and Microsoft® PowerPoint application software, can be used.

Having described and illustrated the principles of our invention with reference to an illustrated embodiment, it will be recognized that the illustrated embodiment can be modified in arrangement and detail without departing from such principles. It should be understood that the programs, processes, or methods described herein are not related or limited to any particular type of computer apparatus, unless indicated otherwise. Various types of general purpose or specialized computer apparatus may be used with or perform operations in accordance with the teachings described herein. Elements of the illustrated embodiment shown in software may be implemented in hardware and vice versa.

In view of the many possible embodiments to which the principles of our invention may be applied, it should be recognized that the detailed embodiments are illustrative only and should not be taken as limiting the scope of our invention. Rather, we claim as our invention all such embodiments as may come within the scope and spirit of the following claims and equivalents thereto.

35

We Claim:

Sub A1

1. In a computer having a display device, a system for displaying a user interface to an operating system as a hypertext multimedia document on the display device, comprising:

a graphical user interface for providing a windowing environment supporting a plurality of windows displayed on the display device according to a front to back order wherein a window towards the front in the order overlaps any windows farther back in the order which are displayed in a same area of the display device; and

a desktop in the windowing environment of the graphical user interface for providing a full-screen view of a hypertext multimedia document displayed back-most in the order.

2. The system of claim 1 comprising:

a plurality of icons in the hypertext multimedia document and operative to launch application programs on user activation.

3. The system of claim 1 comprising:

a plurality of icons in the hypertext multimedia document and operative to activate file system services.

4. The system of claim 1 comprising:

a representation in the hypertext multimedia document of the contents of a folder in a file system of the computer.

Sub A2

5. An operating system shell comprising:

a hypertext multimedia document template;

a preprocessor for processing the template into a hypertext multimedia document having an embedded control object for providing an operating system user interface; and

a viewer for displaying a view of the hypertext document.

6. The operating system shell of claim 5 comprising:

36

Sub A2

displaying the view in a desktop display of a windowing environment on the video screen.

11. The method of claim 9 comprising:

displaying the view in a window of a windowing environment on the video screen.

12. The method of claim 9 comprising:

providing a template having the hypertext data, the object insertion tag, and a pre-processor directive specifying a parameter; and

processing the template to synthesize the hypertext page from the template, the processing including converting the parameter into hypertext data.

Sub A3

13. The method of claim 12 wherein the view of the hypertext page represents a folder of a file system, the method comprising:

storing a configuration file in the folder specifying the template; and

when a user navigates to the folder, identifying the template to use for synthesizing the hypertext page from the configuration file.

14. A file system navigation method comprising:

providing a plurality of templates associated with folders in a file system, the templates having an object insertion tag for embedding a folder user interface control;

when a folder is opened by a user, performing the steps of:

processing the template associated with the opened folder to produce a hypertext page having the object insertion tag; and

displaying a view of the hypertext page with the folder user interface control embedded therein, whereby the user manipulates the folder user interface control to activate file system services relating to the folder.

15. The method of claim 14 comprising:

storing configuration files in at least some of the folders, each configuration file containing data identifying the template associated with the respective folder in which the configuration file is stored;

if the opened folder stores one of the configuration files, determining the template associated with the opened folder from the configuration file stored in the opened folder; and

otherwise, determining a default template is associated with the opened folder.

16. A hypertext page, comprising:

hypertext data; and

an object insertion tag for embedding a user interface control object, said object providing graphical icon-oriented and/or menu driven user interface elements for controlling operating system and/or file system services when the hypertext page is viewed.

Sub A4

~~17. A desktop in a windowing environment of a graphical user interface provided by an operating system shell on a computer, comprising:~~

~~a plurality of graphical icon-oriented and menu-driven user interface elements provided by a control object, the user interface elements being manipulable by a user of the computer to initiate operating system services; and~~

~~a view of a hypertext multimedia document incorporating multi-media enhancements and having the control object embedded therein for integrating the multi-media enhancements with the user interface elements.~~

~~a view container object for providing a desktop display in which the view is embedded.~~

~~7. The operating system shell of claim 5 comprising:~~

~~a view container object for providing an application window in which the view is embedded.~~

~~8. A template for use in a system to synthesize a hypertext multimedia document to provide an operating system user interface, the system having a preprocessor, an embeddable user interface control object, and a hypertext multimedia document viewer, the template comprising:~~

~~hypertext data for incorporating multi-media information into the hypertext multimedia document;~~

~~at least one preprocessor directive for converting parameters into hypertext data; and~~

~~an object insertion tag for embedding the user interface control object into the hypertext multimedia document, whereby the preprocessor processes the template into a hypertext multimedia document which provides an operating system user interface when displayed by the viewer.~~

~~9. A method of providing a graphical user interface to an operating system of a computer having a video screen, comprising:~~

~~providing an operating system user interface control;~~

~~providing a hypertext page having hypertext data for incorporating multi-media enhancements, and an object insertion tag indicative of said control for embedding said control;~~

~~generating a view of the hypertext page incorporating said multi-media enhancements and with said control embedded therein; and~~

~~displaying the view on the video screen.~~

~~10. The method of claim 9 comprising:~~

SAW/jkj 12/8/98 3382-48418 MS821046 EXPRESS MAIL LABEL NO. EM126586705US

COMBINED DECLARATION AND POWER OF ATTORNEY
FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled OPERATING SYSTEM SHELL WITH HYPERTEXT DESKTOP, the specification of which

[X] is attached hereto.

[ ] was filed on __________ as Application No. _______.

[ ] was described and claimed in PCT International Application No. ___________, filed on ___________, and as amended under PCT Article 19 on ___________ (if applicable).

[ ] and was amended on _______________ (if applicable).

[ ] with amendments through ____________ (if applicable).

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, § 1.56. If this is a continuation-in-part application filed under the conditions specified in 35 U.S.C. § 120 which discloses and claims subject matter in addition to that disclosed in the prior copending application, I further acknowledge the duty to disclose material information as defined in 37 CFR § 1.56 which occurred between the filing date of the prior application and the national or PCT international filing date of the continuation-in-part application.

I hereby claim foreign priority benefits under Title 35, United States Code, § 119(a)-(d) of any foreign application(s) for patent or inventor's certificate or of an PCT International application(s) designating at least one country other than the United States of America listed below and have also identified below any foreign application(s) for patent or inventor's certificate or any PCT International application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application(s) on which priority is claimed:

Prior Foreign Applications Priority Claimed

______________ ________ ____________ [ ] Yes [ ] No

(Number) (Country) (Day/Month/Yr. Filed)

I hereby claim the benefit under Title 35, United States Code, § 119(e) of any United States provisional application(s) listed below:

____________________ ______________

(Application No.) (Filing Date)

I hereby claim the benefit under Title 35, United States Code, § 120 of any United States application(s) or § 365(c) of any PCT International application(s) designating the United States, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT International application in the manner provided by the first paragraph of Title 35, United States Code, § 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, § 1.56(a) which occurred between the filing date of the prior application and the national or PCT International filing date of this application:

______________ ______________ ______________________________
(Application No.) (Filing Date) (Status: patented, pending, abandoned)

The undersigned hereby authorizes the U.S. attorney or agent named herein to accept and follow instructions from __________ as to any action to be taken in the Patent and Trademark Office regarding this application without direct communication between the U.S. attorney or agent and the undersigned. In the event of a change in the persons from whom instructions may be taken, the U.S. attorney or agent named herein will be so notified by the undersigned.

I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application, to file a corresponding international application, and to transact all business in the Patent and Trademark Office connected therewith:

Name	Reg. No.	Name	Reg. No.
Kenneth S. Klarquist	16,445	Donald L. Stephens Jr.	34,022
James Campbell	19,978	Stacey C. Slater	36,011
James S. Leigh	20,434	Douglas D. Hancock	35,889
Arthur L. Whinston	19,155	Garth A. Winn	33,220
David P. Petersen	28,106	Stephen A. Wight	37,759
Richard J. Polley	28,107	Joel R. Meyer	37,677
Ramon A. Klitzke II	30,188	Joseph T. Jakubek	34,190
William Y. Conwell	31,943	Mark A. Porter	35,327
Mark L. Becker	31,325	Alan E. Dow	35,123
William D. Noonan	30,878	Mark M. Meininger	32,428
John D. Vandenberg	31,312	Robert F. Scotti	39,830
Patrick W. Hughey	31,169	Gregory V. Bean	36,448
John W. Stuart	24,540	John R. Dawson	39,504

Address all telephone calls to Stephen A. Wight at telephone number (503) 226-7391.

Address all correspondence to:

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP
One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, OR 97204-2988

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of sole or first inventor: Benjamin W. Slivka

Inventor's signature ______________________________ ______________
Date

Residence: Clyde Hill, Washington

Citizenship: USA
Post Office address: 2725 96th Ave., NE, Clyde Hill, WA 98004

Full name of second joint inventor, if any: Teresa Martineau

Inventor's signature ______________________ ____________ Date

Residence: Kirkland, Washington

Citizenship: Canada

Post Office address: 13106 NE 108th St., Kirkland, WA 98033

Full name of third joint inventor, if any: Christopher Ralph Brown

Inventor's signature ______________________ ____________ Date

Residence: Seattle, Washington

Citizenship: United States of America

Post Office address: 1340 E. Interlaken Blvd., Seattle, WA 98102

Full name of fourth joint inventor, if any: George Pitt

Inventor's signature ______________________ ____________ Date

Residence: Redmond, Washington

Citizenship:

Post Office address: One Microsoft Way, Redmond, WA 98052

Full name of fifth joint inventor, if any: Satoshi Nagajima

Inventor's signature ______________________ ____________ Date

Residence: Redmond, Washington

Citizenship:

Post Office address: One Microsoft Way, Redmond, WA 98052

Full name of sixth joint inventor, if any: Sankar Ramasubtamanian

Inventor's signature ______________________ ____________ Date

Residence: Redmond, Washington

Citizenship: India

Post Office address: 17718 NE 104th Wy., Redmond, WA 98052

Full name of seventh joint inventor, if any: Mike Sheldon

Inventor's signature ______________________ ____________ Date

Residence: Redmond, Washington

Citizenship: USA

Post Office address: One Microsoft Way, Redmond, WA 98052

08/761699

APPENDIX A
DESKTOP.HTM

```
<%WinShell = CreateObject("WinShellHTMLPreProc")%>
<!--
 * This file was automatically generated by Microsoft Internet Explorer 4.0   * using
the file <%WinShell.TemplateDirPath%>DESKTOP.HTM  -->


<html>
  <head>
    <title>
      <%WinShell.Title%>
    </title>
  </head>

  <body leftmargin=0 topmargin=0>
    <frameset rows="40,*" frameborder=0 framespacing=0>
      <frame name="tickerframe" scrolling=no
src="<%WinShell.IfProxy("http://iptdweb.microsoft.com/activedesktop/desktop/tickh
ost.htm","%WINDIR%\web\ticker.htm")%>">        <frameset cols="*,240"
frameborder=0 framespacing=0>          <frame name="icons" scrolling=no
src="<%WinShell.ProcessFile("dsk_1.htm")%>">          <FRAME
NAME="infopaneframe"  NORESIZE SCROLLING=NO
SRC="<%WinShell.IfProxy("http://iptdweb.microsoft.com/activedesktop/infopane/inf
opane.htm","%WINDIR%\web\news.htm")%>">        </frameset>
    </frameset>
  </body>

</html>




  <!--
    <frameset rows="64,*" frameborder=no framespacing=0>        <frame
name="dsk2_avi" scrolling=no src= "<%WinShell.TemplateDirPath%>dsk2_avi.htm">
<BUILDTYPE=INTERNAL>
<frame name="dsk2_text" scrolling=yes
src="<%WinShell.IfProxy("http://Themesrv/ie40desk/dsk2_text.htm","%WINDIR%\
web\dsk2_txt.htm")%>"> </BUILDTYPE>
<BUILDTYPE=EXTERNAL>
<frame name="dsk2_text" scrolling=yes
src="<%WinShell.IfProxy("TOK_ADDRESS/ie40desk/dsk2_text.htm","%WINDIR%
\web\dsk2_txt.htm")%>"> </BUILDTYPE>
<BUILDTYPE=RELEASE>
```

```
<frame name="dsk2_text" scrolling=yes
src="<%WinShell.IfProxy("TOK_ADDRESS/ie40desk/dsk2_text.htm","%WINDIR%
\web\dsk2_txt.htm")%>"> </BUILDTYPE>
</frameset>
</body>
```

this stuff needs to be added to reference this file- but for now it is put aside simply for check in -->

APPENDIX B
DSK-1.HTM

```
<%WinShell = CreateObject("WinShellHTMLPreProc")%>
<!--
 * This file was automatically generated by Microsoft Internet Explorer 4.0   * using the file
<%WinShell.TemplateDirPath%>DSK_1.HTM
 -->


<html>

  <style>
    body {
      background: #008080;
      margin-left: 0.13in;
      margin-right: 0.13in;
    }
  </style>


  <body topmargin=12 background="file:<%WinShell.TemplateDirPath%>dskwmark.gif">
 <object classid="clsid:EAB22AC3-30C1-11CF-A7EB-0000C05BAE0B" height=500
width=50>        <param name="Location"         value="shell:Desktop">        <param
name="AlignLeft"        value=1>
      <param name="AutoSize"          value=7>
      <param name="AutoSizePercentage" value=100>
      <param name="AutoArrange"       value=0>
      <param name="NoClientEdge"      value=true>
      <param name="ViewMode"          value=1>
    </object>
  </body>
</html>
```

APPENDIX C
INFOPANE.HTM

```
<HTML>
<HEAD>


<OBJECT ID="timer1"
  CLASSID="clsid:59CCB4A0-727D-11CF-AC36-00AA00A47DD2"
  CODEBASE="#Version=4,70,0,1182"
  TYPE="application/x-oleobject"
  ALIGN=middle
 >
 <PARAM NAME="Interval" VALUE="200">
 <PARAM NAME="Enabled" VALUE="False">
</OBJECT>

<OBJECT id="InfoHelp"
    classid="clsid:3A53F860-D35B-11CF-BB82-00A0C908DBAA"
codebase="http://iptdweb.microsoft.com/activedesktop/infohelp.ocx#Version=4,70,0,1
187"> </OBJECT>

<SCRIPT Language="VBScript">
<!--

Option Explicit

Dim cTicks

Sub Window_onLoad()
  Call DoOnLoad()
End Sub

Sub DoOnLoad()
  'Alert("OnLoad Ipinit")
  InfoHelp.InfoViewName = "Microsoft Catalog"
  If InfoHelp.GetCategories() > 0 Then
    Call InitDone()
  Else
    parent.hiddenframe.location.href="catalog/catdata.htm"       cTicks = 0
    timer1.Interval = 1000
    timer1.Enabled = True
  End If
End Sub

Sub Timer1_timer()
 Dim f
 cTicks = cTicks + 1

 If cTicks >= 15 Then
   timer1.Enabled = False
   MsgBox "Couldn't initialize catalog entries"
 End If

 On Error Resume Next
 f = parent.hiddenframe.FDefaultContentChecked()
 If Err.Number > 0 Then
   Exit Sub
 End If
 On Error Goto 0

 If f Then
   timer1.Enabled = False
   'MsgBox "Content checked after " & cTicks & " ticks!"
   Call InitDone()
 End If
End Sub

Sub InitDone()
   top.tickerframe.ticker1.Start()
   parent.controlframe.location.href="control.htm"
End Sub


-->
</SCRIPT>

</HEAD>

</HTML>
```

APPENDIX C - Page 3

APPENDIX D
NEWS.HTM

```
<!--For Alpha Only - local copy for no proxy users-->

<meta http-equiv="Refresh" content=14400>
<HTML>

<HEAD>

<SCRIPT Language="VBScript">
<!-- '''

Option Explicit

Sub Window_OnLoad()
   Dim Ret
   Ret = OnLineButton.ChangeFont(13, "Arial")
   Ret = OnLineButton.AddButton("Go Online","Switches to live content") End Sub

Sub OnLineButton_Click(x)
   parent.tickerframe.location.href =
"http://iptdweb.microsoft.com/activedesktop/desktop/tickhost.htm"     location.href =
"http://iptdweb.microsoft.com/activedesktop/infopane/infopane.htm" End Sub

"-->
</SCRIPT>

</HEAD>

<BODY LEFTMARGIN=0 TOPMARGIN=0 BGCOLOR=#000000>
<CENTER>
<IMG SRC="IELOGO.GIF" VSPACE=32>
<FONT FACE=TAHOMA SIZE=2 COLOR=gray>
<BR>Press this button
<BR><U>after</U> you are connected to the internet
<BR>to switch to live content.
<BR><BR>
<object id=OnLineButton
  classid="clsid:BEAADBC0-E570-11CF-A9AD-00AA00B92C4D"
  width=72 height=18>
 <PARAM NAME=BordColor VALUE="#000000">
 <PARAM NAME=BackColor VALUE="#D0D0D0">
 <PARAM NAME=ForeColor VALUE="#000000">
 <PARAM NAME=BackHighColor VALUE="#808080">
 <PARAM NAME=ForeHighColor VALUE="#FFFFFF">
 <PARAM NAME=BackClickColor VALUE="#000000">
 <PARAM NAME=ForeClickColor VALUE="#FFFFFF">
</object>
<BR>
<BR>
<BR>
For Alpha use only<br>
</BODY>

</HTML>
```

APPENDIX E
TICKER.HTM

```
<!--For Alpha Only - local copy for no proxy users-->

<meta http-equiv="Refresh" content=14400>
<HTML>
<BODY LEFTMARGIN=0 TOPMARGIN=0 BGCOLOR=#003366 text=gray>
<table width=100%>
<tr>
<td width=200>
<IMG SRC="TCK.GIF">
</td>
<td>
<MARQUEE LOOP=INFINITE>The ticker is not in service at this
time.</MARQUEE> </td>
</tr>
</table>
</BODY>
</HTML>
```

APPENDIX F
TICKHOST.HTM

```
<HTML>

<HEAD>
<TITLE>Microsoft Ticker Sample</TITLE>

</HEAD>

<BODY topmargin=0 leftmargin=0>
<OBJECT
  ID=Ticker1
  CLASSID="clsid:746C0BD4-E002-11CF-9D12-00A0C9034938"

CODEBASE="http://iptdweb.microsoft.com/ActiveDesktop/MSTicker.ocx#Version=
4,70,0,1187"   WIDTH=100%
  HEIGHT=40
  >
</OBJECT>
</BODY>

</HTML>
```

APPENDIX G
SFV2395.TMP

```
<!--
 * This file was automatically generated by Microsoft Internet Explorer 4.0   * using
the file C:\WIN95B\Web\DESKTOP.HTM
 -->


<html>
  <head>
    <title>
      Desktop
    </title>
  </head>

  <body leftmargin=0 topmargin=0>
    <frameset rows="40,*" frameborder=0 framespacing=0>
      <frame name="tickerframe" scrolling=no
src="http://iptdweb.microsoft.com/activedesktop/desktop/tickhost.htm">
<frameset cols="*,240" frameborder=0 framespacing=0>          <frame
name="icons" scrolling=no src="C:\TMP\SFV15143.HTM">          <FRAME
NAME="infopaneframe"  NORESIZE SCROLLING=NO
SRC="http://iptdweb.microsoft.com/activedesktop/infopane/infopane.htm">
</frameset>
    </frameset>
  </body>

</html>




  <!--
    <frameset rows="64,*" frameborder=no framespacing=0>          <frame
name="dsk2_avi" scrolling=no src= "C:\WIN95B\Web\dsk2_avi.htm">
<BUILDTYPE=INTERNAL>
<frame name="dsk2_text" scrolling=yes
src="http://Themesrv/ie40desk/dsk2_text.htm"> </BUILDTYPE>
<BUILDTYPE=EXTERNAL>
<frame name="dsk2_text" scrolling=yes
src="C:\WIN95B\Web\TOK_ADDRESS/ie40desk/dsk2_text.htm"> </BUILDTYPE>
<BUILDTYPE=RELEASE>
<frame name="dsk2_text" scrolling=yes
src="C:\WIN95B\Web\TOK_ADDRESS/ie40desk/dsk2_text.htm"> </BUILDTYPE>
</frameset>
```

```
</body>

    this stuff needs to be added to reference this file- but for now it is put aside simply
for check in    -->
```

APPENDIX G - Page 2

APPENDIX H
SFV15143.HTM

```
<!--
 * This file was automatically generated by Microsoft Internet Explorer 4.0   * using
the file C:\WIN95B\Web\DSK_1.HTM
 -->


<html>

  <style>
    body {
      background: #008080;
      margin-left: 0.13in;
      margin-right: 0.13in;
    }
  </style>


  <body topmargin=12 background="file:C:\WIN95B\Web\dskwmark.gif">
<object classid="clsid:EAB22AC3-30C1-11CF-A7EB-0000C05BAE0B" height=500
width=50>        <param name="Location"         value="shell:Desktop">
<param name="AlignLeft"        value=1>
      <param name="AutoSize"          value=7>
      <param name="AutoSizePercentage" value=100>
      <param name="AutoArrange"       value=0>
      <param name="NoClientEdge"      value=true>
      <param name="ViewMode"          value=1>
    </object>
  </body>
</html>
```

APPENDIX I
DIRECTRY.HTM

```
<%WinShell = CreateObject("WinShellHTMLPreProc")%>
<!--
 * This file was automatically generated by Microsoft Internet Explorer 4.0   * using
the file <%WinShell.TemplateDirPath%>DIRECTRY.HTM  -->


<html>
  <head>
    <title>
      <%WinShell.Title%>
    </title>
  </head>

  <body leftmargin=0 topmargin=0>
    <frameset rows="50, *" frameborder=0 framespacing=0 scrolling=no noresize>
<frame name="top" src="<%WinShell.ProcessFile("dir_1.htm")%>" scrolling=no
noresize>        <frameset cols="*, 165" frameborder=0 framespacing=0 scrolling=no
noresize>          <frame name="icons"
src="<%WinShell.ProcessFile("dir_2.htm")%>" scrolling=yes noresize>
<frame name="text"  src="<%WinShell.ProcessFile("dir_3.htm")%>" scrolling=no
noresize>        </frameset>
    </frameset>
  </body>
</html>
```

APPENDIX J
DIR_1.HTM

```
<%WinShell = CreateObject("WinShellHTMLPreProc")%>
<!--
 * This file was automatically generated by Microsoft Internet Explorer 4.0   * using
the file <%WinShell.TemplateDirPath%>DIR_1.HTM
 -->


<html>

  <style>
    body {
      background: black;
      color: #FF6600;
      font: 0.12in arial;
      margin-left: 0.05in;
      margin-right: 0.05in;
    }
    h1 {
      font: 0.3in Times New Roman;
      color: #006699;
      font-weight: normal;
    }
    a {
      color: #FF6600;
      font-weight: bold;
      margin-left: 0in;
      margin-right: 0in;
    }
  </style>


  <body topmargin=0>
    <table width=100% height=100% cellspacing=0 cellpadding=0 border=0>      <tr
valign=center>
      <td valign=center width=* align=justify>
        <H1>
          <%WinShell.Title%>
        </H1>
      </td>
      <td valign=center align=right>
        <table cellspacing=0 cellpadding=15 border=0>
          <td align=right>
            <a target="icons"
href="<%WinShell.ProcessFile("dir_2.htm")%>">Large</a>
```

```
          <a target="icons"
href="<%WinShell.ProcessFile("dir_4.htm")%>">Small</a>              </td>
        <td align=right>
          <a href="<%WinShell.TemplateDirPath%>..\help\windows.hlp">Help</a>
</td>
       </table>
     </td>
   </tr>
   </table>
 </body>

</html>
```

APPENDIX K
DIR_2.HTM

```
<%WinShell = CreateObject("WinShellHTMLPreProc")%>
<!--
 * This file was automatically generated by Microsoft Internet Explorer 4.0   * using
the file <%WinShell.TemplateDirPath%>DIR_2.HTM
 -->


<html>
  <style>
    body {
      background: #008080;
      margin-left: 0.13in;
      margin-right: 0.13in;
    }
  </style>


  <body topmargin=12
background="file:<%WinShell.TemplateDirPath%>wmark.gif">      <object
classid="clsid:EAB22AC3-30C1-11CF-A7EB-0000C05BAE0B" height=500
width=50>        <param name="Location"
value="<%WinShell.ThisDirPath%>">         <param name="AutoSize"
value=7>
       <param name="AutoSizePercentage" value=100>
       <param name="AutoArrange"        value=0>
       <param name="NoClientEdge"       value=true>
       <param name="ViewMode"           value=1>
     </object>
  </body>
</html>
```

APPENDIX L
DIR_3.HTM

```
<%WinShell = CreateObject("WinShellHTMLPreProc")%>
<!--
 * This file was automatically generated by Microsoft Internet Explorer 4.0   * using
the file <%WinShell.TemplateDirPath%>DIR_3.HTM
 -->


<html>

  <style>
    body {
      background: #006699;
      color: white;
      font: 0.12in arial;
      margin-left: 0.14in;
      margin-right: 0.06in;
    }
    table {
      margin-left: 0in;
      margin-right: 0in;
    }
    hl {
      color: #cc3300;
    }
    p {
    }
    a {
      color: #ff9933;
      font-weight: bold;
    }
  </style>

  <body>
    <p>
    <b>Welcome to <br>Internet Explorer 4.0!</b>
    <p>
    This <b>Alpha</b> version of Microsoft Internet Explorer 4.0 makes using files
 on your hard disk and the Internet completely seamless.     You can now view local
 content in the same way that you view content on the Web.     <p>
    Try it!  Surf the Microsoft web site by clicking any of the links below.     <p>
    <a target="_top" href="http://www.microsoft.com">
      Microsoft Home Page
    </a>
    <br>
    <a target="_top" href="http://www.microsoft.com/misc/whatsnew.htm">
What's New at Microsoft
    </a>
    <br>
    <a target="_top"
href="http://www.microsoft.com/support/products/windows95/windows95.htm">
Product Support
    </a>
  </body>

</html>
```

APPENDIX M
SFV6190.TMP

```
<!--
 * This file was automatically generated by Microsoft Internet Explorer 4.0   * using
the file C:\WINDOWS\Web\DIRECTRY.HTM
 -->


<html>
  <head>
    <title>
      dos
    </title>
  </head>

  <body leftmargin=0 topmargin=0>
    <frameset rows="50, *" frameborder=0 framespacing=0 scrolling=no noresize>
<frame name="top" src="C:\TMP\SFV47916.HTM" scrolling=no noresize>
<frameset cols="*, 165" frameborder=0 framespacing=0 scrolling=no noresize>
<frame name="icons" src="C:\TMP\SFV47932.HTM" scrolling=yes noresize>
<frame name="text"  src="C:\TMP\SFV47945.HTM" scrolling=no noresize>
</frameset>
    </frameset>
  </body>
</html>
```

APPENDIX N
SFV67916.HTM

```
<!--
 * This file was automatically generated by Microsoft Internet Explorer 4.0   * using
the file C:\WINDOWS\Web\DIR_1.HTM
 -->


<html>

  <style>
    body {
      background: black;
      color: #FF6600;
      font: 0.12in arial;
      margin-left: 0.05in;
      margin-right: 0.05in;
    }
    h1 {
      font: 0.3in Times New Roman;
      color: #006699;
      font-weight: normal;
    }
    a {
      color: #FF6600;
      font-weight: bold;
      margin-left: 0in;
      margin-right: 0in;
    }
  </style>


  <body topmargin=0>
    <table width=100% height=100% cellspacing=0 cellpadding=0 border=0>      <tr
valign=center>
      <td valign=center width=* align=justify>
        <H1>
          dos
        </H1>
      </td>
      <td valign=center align=right>
        <table cellspacing=0 cellpadding=15 border=0>
          <td align=right>
            <a target="icons" href="C:\TMP\SFV47917.HTM">Large</a>
            <a target="icons" href="C:\TMP\SFV47931.HTM">Small</a>
</td>
           <td align=right>
            <a href="C:\WINDOWS\Web\..\help\windows.hlp">Help</a>
</td>
         </table>
       </td>
     </tr>
     </table>
   </body>

</html>
```

APPENDIX N - Page 2

APPENDIX O
SFV47917.HTM

```
<!--
 * This file was automatically generated by Microsoft Internet Explorer 4.0   * using
the file C:\WINDOWS\Web\DIR_2.HTM
 -->


<html>
  <style>
    body {
      background: #008080;
      margin-left: 0.13in;
      margin-right: 0.13in;
    }
  </style>


  <body topmargin=12 background="file:C:\WINDOWS\Web\wmark.gif">
<object classid="clsid:EAB22AC3-30C1-11CF-A7EB-0000C05BAE0B" height=500
width=50>      <param name="Location"       value="C:\dos\">       <param
name="AutoSize"        value=7>
      <param name="AutoSizePercentage" value=100>
      <param name="AutoArrange"       value=0>
      <param name="NoClientEdge"      value=true>
      <param name="ViewMode"          value=1>
    </object>
  </body>
</html>
```

APPENDIX P
SFV47945.HTM

```
<!--
 * This file was automatically generated by Microsoft Internet Explorer 4.0   * using
the file C:\WINDOWS\Web\DIR_3.HTM
 -->


<html>

  <style>
    body {
      background: #006699;
      color: white;
      font: 0.12in arial;
      margin-left: 0.14in;
      margin-right: 0.06in;
    }
    table {
      margin-left: 0in;
      margin-right: 0in;
    }
    h1 {
      color: #cc3300;
    }
    p {
    }
    a {
      color: #ff9933;
      font-weight: bold;
    }
  </style>

  <body>
    <p>
    <b>Welcome to <br>Internet Explorer 4.0!</b>
    <p>
    This <b>Alpha</b> version of Microsoft Internet Explorer 4.0 makes using files
on your hard disk and the Internet completely seamless.     You can now view local
content in the same way that you view content on the Web.     <p>
    Try it!  Surf the Microsoft web site by clicking any of the links below.     <p>
    <a target="_top" href="http://www.microsoft.com">
      Microsoft Home Page
    </a>
    <br>
    <a target="_top" href="http://www.microsoft.com/misc/whatsnew.htm">
What's New at Microsoft
    </a>
    <br>
    <a target="_top"
href="http://www.microsoft.com/support/products/windows95/windows95.htm">
Product Support
    </a>
  </body>

</html>
```

APPENDIX Q
HTMLREFW.DOC

HTML Reference

!

<! ... >

Specifies that enclosed text is an author's comment. Any text between the lines will not print. You can include multiple lines of text between the start-tag and end-tag.

Example

```
<!  This line of text, enclosed in an HTML page, will not display.
  This second line of text will not display either.>
```

!DOCTYPE

<!DOCTYPE>

Specifies the version of HTML used in the document. !DOCTYPE is the first element in any HTML document !DOCTYPE is a required element for any HTML 3.2-compliant document.

Example

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
```

A

<A

HREF=*reference*

NAME=*name*

TARGET=*window*

TITLE=*title*

>

Stands for anchor. The A and /A tags enclose text or graphics. The properties of elements that can follow A are applied to the bracketed text or graphic. A is used to attach a hyperlink to text or graphics using the **HREF=**. attributeA is used to specify text or graphics as a named reference using the **NAME=** attribute. Anchors cannot be nested.

HREF=*reference*

Specifies either a destination address or a destination file. A destination address must be in URL format. A destination file must name a file and be in the format of the given file system. If no path or domain name is specified, the file is searched for in the same location as the current document.

NAME=*name*

Specifies a named reference within an HTML document. The name can be referenced by its document and external documents in a hyperlink by prefacing it with the pound sign (#).

TARGET=*window*

Specifies to load the link into the targeted window. This attribute can be used with a frameset where a frame has been named in the FRAME element. The *window* can be one of these values:

window	Specifies to load the link into the targeted window. The *window* must begin with an alphanumeric character to be valid, except for the following four target windows:
_blank	Load the link into a new blank window. This window is not named.
_parent	Load the link into the immediate parent of the document the link is in.
_self	Load the link into the same window the link was clicked in.
_top	Load the link into the full body of the window.

TITLE=*title*

Specifies the title that appears when the hyperlink is selected.

Example

```
<A HREF="http://www.microsoft.com"> This is a link to Microsoft.</A>

<A HREF="home.htm">This is a link to a file called home.htm in the same
directory as this page.</A>

<A TARGET="viewer" HREF="sample.htm">Click here to load the link into
"viewer" window.</A>
```

ADDRESS

<ADDRESS>

Specifies the mailing address. This element typically is used at the bottom of a document. Text is displayed in italics.

Example

```
<ADDRESS>This text will be in italics.</ADDRESS>
```

APPLET

<APPLET
ALIGN=LEFT|CENTER|RIGHT
ALT=*alternateText*
CODE=*appletFile*
CODEBASE=*codebaseURL*
HEIGHT=*pixels*
HSPACE=*pixels* **>**
NAME=*appletInstanceName*

[PARAM NAME =*AttributeName***]**
VSPACE=*pixels*
WIDTH=*pixels***>**

Embeds a Java applet in an HTML document. Requires the end-tag.

ALIGN=*alignment*
Alignment of an object to text.

ALT=*alternateText*
Alternate text for text-only browsers or browsers that do not support Java.

CODE=*appletFile*
The name of the Java applet.

CODEBASE=*codebaseURL*
The base URL of the applet. The directory in which the applet is located.

HEIGHT=*pixels*
Initial height of the applet display area.

HSPACE=*pixels*
Horizontal space.

NAME=*appletInstanceName*
Use **NAME=** to identify an applet to other applets within the HTML page.

NAME=*AttributeName*
Use **NAME=** to pass applet-specific arguments in from an HTML page.

VSPACE=*pixels*
Space in pixels above the applet.

WIDTH=*pixels*
Initial width of the applet display area.

AREA

<AREA
COORDS=*coords*
HREF=*url*
NOHREF
SHAPE=*shape-type*
TARGET=*window* **>**

Specifies the shape of a "hot spot" in a client-side image map.

COORDS=*coords*
Specifies coordinates that define the hot spot's shape.

HREF=*url*
Specifies the destination of the hot spot.

NOHREF
Indicates that clicks in this region should cause no action.

SHAPE=*shape-type*

Denotes the type of shape. The *shape-type* can be one of these values:

RECT	Rectangle. Takes four coordinates: *x1*, *y1*, *x2*, and *y2*.
RECTANGLE	Rectangle. Takes four coordinates: *x1*, *y1*, *x2*, and *y2*.
CIRC	Circle. Takes three coordinates: *centerx*, *centery*, and *radius*.
CIRCLE	Circle. Takes three coordinates: *centerx*, *centery*, and *radius*.
POLY	Polygon. Takes three or more pairs of coordinates denoting a polygonal region.
POLYGON	Polygon. Takes three or more pairs of coordinates denoting a polygonal region.

TARGET=*window*

Specifies to load the link into the targeted window. The *window* can be one of these values:

window	Specifies to load the link into the targeted window. The *window* must begin with an alphanumeric character to be valid, except for the following four target windows:
_blank	Load the link into a new blank window. This window is not named.
_parent	Load the link into the immediate parent of the document the link is in.
_self	Load the link into the same window the link was clicked in.
_top	Load the link into the full body of the window.

Examples

```
<AREA SHAPE="RECT" COORDS="50, 25, 150, 125" HREF="http://www.sample.com">

<AREA SHAPE="RECT" COORDS="50, 25, 150, 125" NOHREF>

<AREA TARGET="viewer" HREF="sample.htm" SHAPE="CIRCLE" COORDS="50, 25, 150,
125">
```

B

<B>

Renders text in bold.

Example

```
<B>Displayed in a bold typeface.</B>
```

BASE

<BASE
HREF=*url*
TARGET=*window*>

Specifies the document's URL.

APPENDIX Q - Page 4

HREF=*url*

Specifies the document's full URL in case the document gets read out of context and the reader wants to refer to the original.

TARGET=*window*

Specifies to load all of the links on the page into the targeted window. This can be overridden by specifying a different target attribute for a specific link. The *window* can be one of these values:

window	Specifies to load the link into the targeted window. The *window* must begin with an alphanumeric character to be valid, except for the following four target windows:
_blank	Load the link into a new blank window. This window is not named.
_parent	Load the link into the immediate parent of the document the link is in.
_self	Load the link into the same window the link was clicked in.
_top	Load the link into the full body of the window.

Examples

```
<BASE HREF="http:// www.sample.com/hello.htm">
```

```
<BASE HREF="http:// www.sample.com/hello.htm" TARGET="viewer">
```

BASEFONT

<BASEFONT
COLOR=*color*
NAME=*name*
SIZE=*n***>**

Sets the base font value. This value will be used as a default for any text not formatted with a style sheet or using the FONT element.

COLOR=*color*

Specifies the color of the base font.

NAME=*name*

Specifies the name of the base font.

SIZE=*n*

Specifies the size of the base font. The *n* can be between 1 and 7 inclusive; default is 3; 7 is largest. Throughout the document, relative font size settings (for example, <FONT SIZE=+3>) are set according to this.

Example

```
<BASEFONT SIZE=3> This sets the base font size to 3.
<FONT SIZE=+4> Now the font size is 7.
<FONT SIZE=-1> Now the font size is 2.
```

BGSOUND

<BGSOUND
SRC=*url*
LOOP=*n* **>**

Adds background sounds or "soundtracks" to a page. Sounds can either be samples (.wav or .au format) or MIDI format.

SRC=*url*

Specifies the address of a sound to be displayed.

LOOP=*n*

Specifies how many times a sound will loop when activated. If *n*=-1, or if **LOOP**=INFINITE is specified, it will loop indefinitely.

BIG

<BIG>

Makes text one size larger.

Example

```
<BIG>This text is larger.</BIG>
```

BLOCKQUOTE

<BLOCKQUOTE>

Indents both left and right margins. Used to set apart quotations in text.

Example

```
<P>He said,
<BLOCKQUOTE>"Hi there!" </BLOCKQUOTE>
```

BODY

<BODY
BACKGROUND=*url*
BGCOLOR=*color*
BGPROPERTIES=FIXED
LINK=*color*
TEXT=*color*
TOPMARGIN=*n*
VLINK=*color***>**

APPENDIX Q - Page 6

Specifies the beginning and end of the document body. This element also allows you to set the background image, the background color, the link colors, and the top and left margins of the page.

BACKGROUND=*url*

Specifies a background picture. The picture is tiled behind the text and graphics on the page.

BGCOLOR=*color*

Sets the background color of the page. The *color* can be either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

BGPROPERTIES=FIXED

Specifies a watermark, which is a background picture that does not scroll.

LEFTMARGIN=*n*

Specifies the left margin for the entire body of the page and overrides the default margin. If set to zero, the left margin will be exactly on the left edge.

LINK=*color*

Sets the color of hyperlinks that have not yet been visited. The *color* can be either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

TEXT=*color*

Sets the color of text on the page. The *color* can be either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

TOPMARGIN=*n*

Specifies the margin for the top of the page and overrides the default margin. If set to zero, the top margin will be on the precise top edge.

VLINK=*color* or *colorname*

Sets the color of hyperlinks that have already been visited. The *color* can be either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

Examples

The HTML used to insert the background image of this page is:

```
<BODY BACKGROUND="/ie/images/watermrk.gif" BGPROPERTIES=FIXED
BGCOLOR=#FFFFFF TEXT=#000000 LINK=#ff6600 VLINK=#330099>


<HTML> <BODY>Here's a Web page!</BODY></HTML>
```

BR

<BR
CLEAR=LEFT|RIGHT|ALL>

Inserts a line break.

CLEAR=LEFT|RIGHT|ALL

Inserts vertical space so that the next text displayed will be past left- or right-aligned "floating" images. The *align-type* can be LEFT, RIGHT, or ALL.

- LEFT inserts space so that the next text appears aligned with the left margin directly below a left-aligned floating image.
- RIGHT inserts space so that the next text appears aligned with the right margin directly below a right-aligned floating image.
- ALL places the next text past all floating images.

CAPTION

<CAPTION
ALIGN=TOP|BOTTOM>

Specifies a caption for a table. This element is valid only within the TABLE element. The end-tag is required.

ALIGN=TOP|BOTTOM

Sets the alignment of the caption within the table. The *align-type* can be LEFT, RIGHT, TOP, or BOTTOM. By default, the caption is centered and at the bottom of the table.

Example

```
<TABLE>
<CAPTION ALIGN=BOTTOM>
This caption will appear centered below the table.
</CAPTION>
<TR>
    ....
</TR>
</TABLE>
```

CENTER

<CENTER>

Centers text and images.

Example

```
<CENTER>Hi there!</CENTER>
```

CITE

<CITE>

Indicates a citation. Refers to a book, paper, or other published source material.

Example

```
<CITE>Book Title.</CITE>
```

CODE

<CODE>

Specifies a code sample. Renders text in a small font. (If no font face is specified, the font used is fixed-width.)

Example

```
<CODE>Here is some text in a small, fixed-width font.</CODE>
```

COL

<COL
ALIGN=LEFT|CENTER|RIGHT
SPAN=*n*>

Sets the properties of one or more columns. Use this element in conjunction with a COLGROUP element to set the properties of a column within a group of columns.

ALIGN=LEFT|CENTER|RIGHT
: Specifies the text alignment in cells within the column. The *align-type* can be LEFT, CENTER, or RIGHT.

SPAN=*n*
: Sets the number of consecutive columns for which the properties are set.

This element is valid only within a table. The end-tag is not required and is not recommended.

The properties specified by the COL element always override the properties specified by the preceding COLGROUP element

Example

```
<TABLE>
<COLGROUP>
    <COL ALIGN=RIGHT>
    <COL ALIGN=LEFT>
<COLGROUP>
    <COL ALIGN=CENTER>
<TBODY>
    <TR>
    <TD>This is the first column in the group and is right-aligned.</TD>
    <TD>This is the second column in the group and is left-aligned.</TD>
    <TD>This column is in a new group and is centered.</TD>
    </TR>
</TABLE>
```

COLGROUP

<COLGROUP
ALIGN=LEFT|CENTER|RIGHT

SPAN=*n*>

Sets the properties of one or more columns.

ALIGN=LEFT|CENTER|RIGHT

Specifies the alignment of text in the cells in the column(s). The *align-type* can be LEFT, CENTER, or RIGHT.

SPAN=*n*

Sets the number of consecutive columns that are in the group and for which the properties are set.

This element is valid only within a table. The end-tag is not required and is not recommended.

If the columns in a group of columns require varying properties, use COLGROUP in conjunction with one or more COL elements to individually set the properties for the columns.

This element affects how rules are drawn within a table when groups are specified with the RULES= attribute in the TABLE element. In this case, vertical rules are drawn between column groups rather than between individual columns.

Example

```
<TABLE>
<COLGROUP ALIGN=RIGHT>
<COLGROUP SPAN=2 ALIGN=LEFT>
<TBODY>
    <TR>
    <TD>This column is in the first group and is right-aligned.</TD>
    <TD>This column is in the second group and is left-aligned.</TD>
    <TD>This column is in the second group and is left-aligned.</TD>
    </TR>
</TABLE>
```

COMMENT

<COMMENT>

Indicates a comment. The text between the elements is ignored, unless it contains HTML code.

Example

```
<COMMENT>This won't be printed.</COMMENT>
```

DD

<DD>

Specifies a definition in a definition list. Indicates that the text is a definition of a term, and should therefore be displayed in the right-hand column of a definition list.

Example

```
<DIV>
This text represents a section.
</DIV>

<DIV ALIGN=CENTER>
This text represents another section.
</DIV>
```

DL

<DL>

Specifies that the following block is a definition list, that is, an automatically formatted two-column list with terms on the left and their definitions on the right.

Example

```
<DL>
<DT>Cat
<DD>A furry, cute animal that purrs and likes milk.
<DT>Lizard
<DD>A weird desert animal with a long tongue.
</DL>
```

DT

<DT>

Specifies a term in a definition list. Indicates that the text is a term to be defined, and should therefore be displayed in the left-hand column of a definition list.

Example

```
<DL> <DT>Cat<DD>A furry, cute animal that purrs and likes milk.
<DT>Lizard<DD>A weird desert animal with a long tongue.</DL>
```

EM

<EM>

Emphasizes text, usually by rendering it in italics.

Example

```
<EM>This text will be in italics.</EM>
```

EMBED

```
<DL> <DT>Cat<DD>A furry, cute animal that purrs and likes milk.
<DT>Lizard<DD>A weird desert animal with a long tongue.</DL>
```

DFN

<DFN>

Specifies a definition. Formats a term for its first appearance in a document.

Example

```
<DFN>HTML stands for hypertext markup language.</DFN>
```

DIR

<DIR>

Denotes a directory list. Specifies that the following block consists of individual items, each beginning with an LI element and none containing more than 20 characters, that should be displayed in columns.

Example

```
<DIR> <LI>Art
<LI>History
<LI>Literature
<LI>Sports
<LI>Entertainment
<LI>Science
</DIR>
```

DIV

<DIV
ALIGN=LEFT|CENTER|RIGHT|JUSTIFY
CLASS=*container type*
/DIV>

Represents different kinds of containers, for example, chapter, section, abstract, or appendix, when used with the **CLASS** attribute. DIV allows the enclosed group of elements to be given a distinctive style.

ALIGN=LEFT|CENTER|RIGHT|JUSTIFY

Specifies the default horizontal alignment for the contents of the DIV element, when DIV is used with an **ALIGN** attribute. This is needed for compatibility with deployed browsers and may be overridden by style sheets.

CLASS=*container type*

Represents different kinds of containers, for example, chapter, section, abstract, or appendix. DIV allows the enclosed group of elements to be given a distinctive style.

Example

<EMBED
HEIGHT=*height of object*
NAME=*programmatic name*
OPTIONAL PARAM="value" ... OPTIONAL PARAM=
PALETTE=*foreground|background*
SRC=*location of object*
WIDTH=*width of object***>**

Indicates an embedded object. OBJECT is the preferred element for inserting objects, but EMBED is included for backward compatibility with earlier HTML documents. See OBJECT.

HEIGHT=*size of object*
The height, in pixels, of the object on the page.

NAME=*programmatic name*
The name used by other objects or elements to refer to this object.

OPTIONAL PARAM=*value*
Specifies any parameters that are specific to the object.

PALETTE=*foreground|background*;
Sets the color palette to the foreground or background color.

SRC=*data to object*
The name of any source data input to the object.

WIDTH=*size of object*
The width of the object, in pixels, on the page.

Example

```
<EMBED SRC=&quot;MyMovie.AVI&quot; WIDTH=100 HEIGHT=250 AUTOSTART=TRUE
PLAYBACK=FALSE&gt;</code></font></font></pre>
```

FONT

<FONT
SIZE=*n*
FACE=*name*
COLOR=*color***>**

Sets the size, font, and color of text.

SIZE=*n*
Specifies font size between 1 and 7 (7 is largest). A plus or minus before the number indicates a size relative to the current BASEFONT setting. Relative font sizes are not cumulative, so putting two <FONT SIZE="+1"> elements in a row does not result in the font size being increased by 2.

FACE="*name* [,*name2*[,*name3*]]"
Sets the font. A list of font names can be specified. If the first font is available on the system, it will be used; otherwise, the second will be tried, and so on. If none are available, a default font will be used.

COLOR=*color*

Sets font color. The *color* can be either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

FORM

<FORM
ACTION=*url*
METHOD=*get-post*
TARGET=*window*>

Denotes a form.

ACTION=*url*

Specifies the address to be used to carry out the action of the form. If none is specified, the base URL of the document is used.

METHOD=*get-post*

Indicates how the form data should be sent to the server. The *get-post* can be one of these values:

GET	Appends the arguments to the action URL and opens it as if it were an anchor.
POST	Sends the data via an HTTP post transaction.

TARGET=*window*

Specifies to load the results of the form submission into the targeted window. The *window* can be one of these values:

window	Specifies to load the link into the targeted window. The *window* must begin with an alphanumeric character to be valid, except for the following four target windows:
_blank	Load the link into a new blank window. This window is not named.
_parent	Load the link into the immediate parent of the document the link is in.
_self	Load the link into the same window the link was clicked in.
_top	Load the link into the full body of the window.

Example

```
<FORM TARGET="viewer" ACTION="http://www.sample.com/bin/search">
    ...
</FORM>
```

FRAME

<FRAME
ALIGN=LEFT|CENTER|RIGHT|TOP|BOTTOM
FRAMEBORDER=1|0
MARGINHEIGHT=*height*
MARGINWIDTH=*width*
NAME=*name*
SCROLLING=yes|no
SRC=*address*

Defines a single frame in a frameset. There is no matching end-tag as this is not a container.

ALIGN=LEFT|CENTER|RIGHT

. Sets the alignment of the frame or of the surrounding text. The *align-type* can be one of these values:

TOP	Surrounding text is aligned with the top of the frame.
MIDDLE	Surrounding text is aligned with the middle of the frame.
BOTTOM	Surrounding text is aligned with the bottom of the frame.
LEFT	The frame is drawn as a left-flush "floating frame," and text flows around it.
RIGHT	The frame is drawn as a right-flush "floating frame," and text flows around it.

FRAMEBORDER=1|0

Renders a 3-D edge border around the frame. 1 (default) inserts a border. 0 displays no border.

MARGINHEIGHT=*height*

Controls the margin height for the frame, in pixels.

MARGINWIDTH=*width*

Controls the margin width for the frame, in pixels.

NAME=*name*

Provides a target name for the frame.

NORESIZE

Prevents the user from resizing the frame.

SCROLLING=*yes|no*

Creates a scrolling frame.

SRC=*address*

Displays the source file for the frame.

Example

```
<FRAME FRAMEBORDER=0 SCROLLING=NO SRC="sample.htm">
```

FRAMESET

<FRAMESET
COLS=*col-widths*
FRAMEBORDER=1|0
FRAMESPACING=*spacing*
ROWS=*row-heights*

Hosts the FRAME, **FRAMESET**, and NOFRAMES elements.

COLS=*col-widths*

Creates a frame document with columns. You can specify the column dimensions by

APPENDIX Q - Page 15

percentage (%), pixels, or a relative size (*).

FRAMEBORDER=1|0

Provides the option to display or not display a 3-D border for a frame. 1 (default) sets a frame border. 0 displays no border.

FRAMESPACING=*spacing*

Creates additional space between frames, in pixels.

ROWS=*row-heights*

Creates a frame document with rows. You can specify the row dimensions by percentage (%), pixels, or a relative size (*).

The **FRAMEBORDER=** and **FRAMESPACING=** attributes are inherited from any containing FRAMESET element, which means you need only set the attribute on the single, outermost FRAMESET tag to affect all FRAME tags on that page.

Example

```
<FRAMESET SCROLLING=YES COLS="25%, 50%, *">
    <FRAME SRC="contents.htm">
    <FRAME SRC="info.htm">
    <FRAME SCROLLING=NO SRC="graphic.htm">
</FRAMESET>
```

Hn

<H*n*
ALIGN=LEFT|CENTER|RIGHT>

Renders text in heading style. Use H1 through H6 to specify different sizes and styles of heading.

n

Sets the section level. This is an integer from 1 to 6.

ALIGN=LEFT|CENTER|RIGHT

Sets the alignment of header text. *Align-type* can be LEFT, CENTER, or RIGHT. LEFT is default.

The end-tag is required.

Example

```
<H1>Welcome to Internet Explorer!</H1>
```

HEAD

<HEAD>

Marks the HTML document heading.

The end-tag is not required.

Example

```
<HEAD>
<TITLE>A Simple Document</TITLE>
</HEAD>
```

HR

<HR
ALIGN=LEFT|CENTER|RIGHT
COLOR=*color*
NOSHADE
SIZE=*n*
WIDTH=*n*>

Draws a horizontal rule.

ALIGN=LEFT|CENTER|RIGHT
: Draws the rule left-aligned, right-aligned, or centered. The *align-type* can be LEFT, RIGHT, or CENTER.

COLOR=*color*
: Sets the color of the rule. The *color* can be either a hexadecimal, red-green-blue color value or a predefined color name. See Color.

NOSHADE
: Draws the rule without 3-D shading.

SIZE=*n*
: Sets the height of the rule, in pixels.

WIDTH=*n*
: Sets the width of the rule, either in pixels or as a percentage of window width. To specify a percentage, the *n* must end with the percent (%) sign.

Example

```
<HR SIZE=5 WIDTH=80% NOSHADE>
```

HTML

<HTML>

Denotes the file as an HTML document.

This element has no attributes.

Example

APPENDIX Q - Page 17

```
<HTML>
<BODY>
<P>This is an HTML document.
</BODY>
</HTML>
```

I

<I>

Renders text in italics.

Example

```
<I>This text will be in italics.</I>
```

IFRAME

<IFRAME
ALIGN=LEFT|CENTER|RIGHT
FRAMEBORDER=1|0
HEIGHT=*height*
MARGINHEIGHT=*height*
MARGINWIDTH=*width*
NAME=*name*
SCROLLING=yes|no
SRC=*address*
WIDTH=*width*
/IFRAME>

Defines a floating frame. This element requires the end-tag.

ALIGN=LEFT|CENTER|RIGHT

Sets the alignment of the frame or of the surrounding text. The *align-type* can be one of these values:

TOP	Surrounding text is aligned with the top of the frame.
MIDDLE	Surrounding text is aligned with the middle of the frame.
BOTTOM	Surrounding text is aligned with the bottom of the frame.
LEFT	The frame is drawn as a left-flush "floating frame," and text flows around it.
RIGHT	The frame is drawn as a right-flush "floating frame," and text flows around it.

FRAMEBORDER=1|0

Renders a 3-D edge border around the frame. 1 (default) inserts a border. 0 displays no border.

HEIGHT

Controls the height (in pixels) of the floating frame.

MARGINHEIGHT=*height*

Controls the margin height for the frame, in pixels.

MARGINWIDTH=*width*

Controls the margin width for the frame, in pixels.

NAME=*name*

Provides a target name for the frame.

NORESIZE

Prevents the user from resizing the frame.

SCROLLING=yes|no

Creates a scrolling frame.

SRC=*address*

Displays the source file for the frame.

WIDTH=width

Controls the width (in pixels) of the floating frame.

Example

```
<IFRAME FRAMEBORDER=0 SCROLLING=NO SRC="sample.htm"></IFRAME>
```

IMG

<IMG
ALIGN=LEFT|CENTER|RIGHT
ALT=*text*
BORDER=*n*
CONTROLS
DYNSRC=*url*
HEIGHT=*n*
HSPACE=*n*
ISMAP
LOOP=*n*
SRC=*address*
START=*start-event*
USEMAP=*map-name*
VSPACE=*n*
WIDTH=*n***>**

Inserts an image.

ALIGN=LEFT|CENTER|RIGHT

Sets the alignment of the image or of the surrounding text. The *align-type* can be one of these values:

TOP	Surrounding text is aligned with the top of the image.
MIDDLE	Surrounding text is aligned with the middle of the image.
BOTTOM	Surrounding text is aligned with the bottom of the image.

LEFT	The picture is drawn as a left-flush "floating image," and text flows around it.
RIGHT	The picture is drawn as a right-flush "floating image," and text flows around it.

ALT=*text*

Specifies text that will be displayed in place of the picture if Show Pictures is turned off.

BORDER=*n*

Specifies the size of a border to be drawn around the image. If the image is a hyperlink, the border is drawn in the appropriate hyperlink color. If the image is not a hyperlink, the border is invisible.

CONTROLS

If a video clip is present, displays a set of controls under the clip.

DYNSRC=*url*

Specifies the address of a video clip or VRML world to be displayed in the window. Stands for Dynamic Source.

HEIGHT=*n*

Along with **WIDTH=**, specifies the size at which the picture is drawn. If the picture's actual dimensions differ from those specified, the picture is stretched to match what's specified. Internet Explorer also uses this to draw a placeholder of appropriate size for the picture before it is loaded.

HSPACE=*n*

Along with **VSPACE=**, specifies margins for the image. Similar to **BORDER=**, except the margins are not painted with color when the image is a hyperlink.

ISMAP

Identifies the picture as a server-side image map. Clicking the picture transmits the coordinates of the click back to the server, triggering a jump to another page.

LOOP=*n*

Specifies how many times a video clip will loop when activated. If *n*=-1, or if **LOOP=**INFINITE is specified, it will loop indefinitely.

SRC=*address*

Specifies the address of the picture to insert.

START=*start-event*

Specifies when the file specified by the **DYNSRC=** attribute should start playing. The *start-event* can be one of these values:

FILEOPEN	Start playing as soon as the file is done opening. This is the default.
MOUSEOVER	Start playing when the user moves the mouse pointer over the animation.

Both values can be set but must be separated with a comma.

USEMAP=*map-name*

Identifies the picture as a client-side image map and specifies a MAP to use for acting on the user's clicks.

VSPACE=*n*

Along with **HSPACE=**, specifies margins for the image. Similar to **BORDER=**, except the margins are not painted with color when the image is a hyperlink.

WIDTH=*n*

Along with **HEIGHT=**, specifies the size at which the picture is drawn. If the picture's actual dimensions differ from those specified, the picture is stretched to match what's specified. Internet Explorer also uses this to draw a placeholder of appropriate size for the picture before it is loaded.

INPUT

<INPUT
ALIGN=LEFT|CENTER|RIGHT
[CHECKED|]
MAXLENGTH=*length*
NAME=*name*
SIZE=*size*
SRC=*address*
TYPE=*type*
VALUE=*value***>**

Specifies a form control.

ALIGN=LEFT|CENTER|RIGHT

Specifies how the next line of text will be aligned with the image. Used when **TYPE=**IMAGE. The *align-type* can be TOP, MIDDLE, or BOTTOM.

CHECKED

Sets a check box or radio button to "selected" when the form first loads.

MAXLENGTH=*length*

Indicates the maximum number of characters that can be entered into a text control.

NAME=*name*

Specifies the name of the control.

SIZE=*size*

Specifies the size of the control (in characters). For TEXTAREA-type controls, both height and width can be specified using this format: "*width,height*".

SRC=*address*

Specifies the address of the image to be used. Used when **TYPE=**IMAGE.

TYPE=*type*

Specifies what type of control to use. The *type* can be one of these values:

CHECKBOX

Used for simple Boolean attributes or for attributes that can take multiple values at the same time. It is represented by a number of check box fields, each of which has the same name. Each selected check box generates a separate name/value pair in the submitted data, even if this results in duplicate names. The default value for check boxes is "on."

HIDDEN	No field is presented to the user, but the content of the field is sent with the submitted form. This value can be used to transmit state information about client/server interaction.
IMAGE	An image field that you can click, causing the form to be immediately submitted. The coordinates of the selected point are measured in pixel units from the upper-left corner of the image, and are returned (along with the other contents of the form) in two name/value pairs. The x-coordinate is submitted under the name of the field with ".x" appended, and the y-coordinate is submitted under the name of the field with ".y" appended. Any **VALUE** attribute is ignored. The image itself is specified by the **SRC** attribute, exactly as for the IMAGE element.
PASSWORD	The same as the TEXT attribute, except that text is not displayed as the user enters it.
RADIO	Used for attributes that accept a single value from a set of alternatives. Each radio button field in the group should be given the same name. Only the selected radio button in the group generates a name/value pair in the submitted data. Radio buttons require an explicit **VALUE** attribute.
RESET	A button that, when clicked, resets the form's fields to their specified initial values. The label to be displayed on the button can be specified just as for the SUBMIT button.
SUBMIT	A button that, when clicked, submits the form. You can use the **VALUE** attribute to provide a non-editable label to be displayed on the button. The default label is application-specific. If a SUBMIT button is clicked in order to submit the form, and that button has a **NAME** attribute specified, that button contributes a name/value pair to the submitted data. Otherwise, a SUBMIT button makes no contribution to the submitted data.
TEXT	Used for a single-line text-entry field. Use in conjunction with the **SIZE** and **MAXLENGTH** attributes.

This default control type is TEXT.

VALUE=*value*

For textual/numerical controls, specifies the default value of the control. For Boolean controls, specifies the value to be returned when the control is turned on.

Example

```
<FORM ACTION="http://intranet/survey" METHOD=POST>
<P>Name
<BR><INPUT NAME="CONTROL1" TYPE=TEXT VALUE="Your Name">
<P>Password
<BR><INPUT TYPE="PASSWORD" NAME="CONTROL2">
<P>Color
<BR><INPUT TYPE="RADIO" NAME="CONTROL3" VALUE="0" CHECKED>Red
<INPUT TYPE="RADIO" NAME="CONTROL3" VALUE="1">Green
<INPUT TYPE="RADIO" NAME="CONTROL3" VALUE="2">Blue
<P>Comments
<BR><INPUT TYPE="TEXTAREA" NAME="CONTROL4" SIZE="20,5" MAXLENGTH="250">
<P><INPUT NAME="CONTROL5" TYPE=CHECKBOX CHECKED>Send receipt
<P><INPUT TYPE="SUBMIT" VALUE="OK"><INPUT TYPE="RESET" VALUE="Reset">
</FORM>
```

ISINDEX

<ISINDEX
ACTION=*url*
PROMPT=*prompt-text***>**

Indicates the presence of a searchable index.

ACTION=*url*

Specifies the gateway program to which the string in the text box should be passed.

PROMPT=*prompt-text*

Specifies a prompt to be used instead of the default prompt.

If the **PROMPT=** attribute is not used, the element displays the following message, followed by a text box:

"You can search this index. Type the keyword(s) you want to search for:"

When the user enters text and presses ENTER, that text is posted back to the page's URL as a query.

Example

```
<ISINDEX "http://intranet/search" PROMPT="Type keywords here.">
```

KBD

<KBD>

Text to be entered at the keyboard. Renders text in fixed-width and bold type.

Example

```
<KBD>The user should enter this text.</KBD>
```

Source

HTML 2

LI

<LI
TYPE=*order-type*
VALUE=*n***>**

Denotes one item of a list. In a DIR , MENU ,OL or UL block, denotes a new list item.

TYPE=*order-type*

Changes the style of an ordered list. The *order-type* can be one of these values:

A	Use large letters.
a	Use small letters.
I	Use large Roman numerals.
i	Use small Roman numerals.
1	Use numbers.

VALUE=*n*

Changes the count of ordered lists as they progress.

Example

```
<DIR> <LI>Art
<LI>History
<LI>Literature
<LI>Sports
<LI>Entertainment
<LI>Science</DIR>
```

LINK

<LINK
HREF=*URL***>**
REL=*forward link type*
REV=*reverse link type*
TITLE=*title*
TYPE=*type*

Establishes a hierarchical organization for navigating between documents. The LINK element must reside within the HEAD element. The HEAD element may contain several LINK elements.

HREF=*URL*

Specifies the URL that has a relationship to the current document.

REL=*forward link type*

Specifies the forward link type, that is, the type of document to which the link is being made. Note that link type "stylesheet" is hereby defined to signify that the associated LINK element specifies a link to a style sheet that may be applied to the HTML document containing the LINK element.

REV=*reverse link type*

TITLE=*title*

Indicates an advisory title string. The character string supplied with this attribute is recommended for use in building a menu of alternative styles.

TYPE=*type*

Specifies the Internet media type and associated parameters for the linked style sheet. This allows the user agent to disregard style sheets in unsupported notations, without the need to first make a remote query across the network.

Example

```
<LINK HREF="http://www.microsoft.com/newdocnewdoc.htm">
```

LISTING

<LISTING>

Renders text in fixed-width type.

Example

```
<LISTING>Here's some plain text.</LISTING>
```

MAP

<MAP
NAME=*name***>**

Specifies a collection of hot spots for a client-side image map.

NAME=*name*

Gives the MAP a name so it can be referred to later. See below for an example of a client-side image map.

Example

```
<MAP NAME="map1">
     <AREA ... >
    <AREA ... >
</MAP>
```

MARQUEE

<MARQUEE
ALIGN=LEFT|CENTER|RIGHT
BEHAVIOR=*type*
BGCOLOR=*color*
DIRECTION=*direction*
HEIGHT=*n*
HSPACE=*n*

LOOP=*n*
SCROLLAMOUNT=*n*
SCROLLDELAY=*n*
VSPACE=*n*
WIDTH=*n*>

Creates a scrolling text marquee.

ALIGN=LEFT|CENTER|RIGHT

Specifies how the surrounding text should align with the marquee. The *align-type* can be one of these values:

TOP	Surrounding text aligns with the top of the marquee.
MIDDLE	Surrounding text aligns with the middle of the marquee.
BOTTOM	Surrounding text aligns with the bottom of the marquee.

BEHAVIOR=*type*

Specifies how the text should behave. The *type* can be one of these values:

SCROLL	Start completely off one side, scroll all the way across and completely off, and then start again. This is the default.
SLIDE	Start completely off one side, scroll in, and stop as soon as the text touches the other margin.
ALTERNATE	Bounce back and forth within the marquee.

BGCOLOR=*color*

Specifies a background color for the marquee. The *color* can be either a hexadecimal number (optionally preceded by a number (#) sign) specifying a red-green-blue color value, or a predefined color name as described in. Color

DIRECTION=*direction*

Specifies in which direction the text should scroll. The *direction* can be LEFT or RIGHT. The default is LEFT, which means scrolling to the left from the right.

HEIGHT=*n*

Specifies the height of the marquee, either in pixels or as a percentage of the screen height. To specify a percentage, the *n* must end with a percent (%) sign.

HSPACE=*n*

Specifies left and right margins for the outside of the marquee, in pixels.

LOOP=*n*

Specifies how many times a marquee will loop when activated. If *n*=-1, or if **LOOP=**INFINITE is specified, it will loop indefinitely.

SCROLLAMOUNT=*n*

Specifies the number of pixels between each successive draw of the marquee text.

SCROLLDELAY=*n*

Specifies the number of milliseconds between each successive draw of the marquee text.

VSPACE=*n*

Specifies top and bottom margins for the outside of the marquee, in pixels.

WIDTH=*n*

Sets the width of the marquee, either in pixels or as a percentage of the screen width. To specify a percentage, the *n* must end with a percent (%) sign.

Example

```
<MARQUEE DIRECTION=RIGHT BEHAVIOR=SCROLL SCROLLAMOUNT=10
SCROLLDELAY=200>This is a scrolling marquee.</MARQUEE>
```

MENU

<MENU>

Denotes a list of items. Specifies that the following block consists of individual items, each beginning with an LI element.

Example

```
<MENU>
<LI>This is the first item in the menu.
<LI>And this is the second item in the menu.
</MENU>
```

META

<META
HTTP-EQUIV=*response*
CONTENT=*description*
NAME=*description*
URL=*url*>

Provides information about an HTML document to browsers, search engines, and other applications.

HTTP-EQUIV=*response*

Binds the element to an HTTP response header. This information is then used based on the application reading the header. See the examples that follow.

CONTENT=*description*

Defines the meta-information content to be associated with the given name or HTTP response header. Can be used with **URL=** and a date and time specification to reload a document at a specified interval. See the Author's Guide section "Client Pull" or the examples that follow.

NAME=*description*

Contains a description of the document.

URL=*url*

Indicates the document's URL.

Examples

If the document contains:

```
<META HTTP-EQUIV="Expires"
      CONTENT="Tue, 04 Dec 1996 21:29:02 GMT">
<meta http-equiv="Keywords" CONTENT="HTML, Reference">
<META HTTP-EQUIV="Reply-to"
      content="anybody@microsoft.com">
<Meta Http-equiv="Keywords" CONTENT="HTML Reference Guide">
```

then the server may include the following header fields:

Expires: Tue, 04 Dec 1996 21:29:02 GMT

Keywords: HTML, Reference

Reply-to: anybody@microsoft.com

as part of the HTTP response to a GET or HEAD request for that document.

```
<HTML>
<HEAD>
<META HTTP-EQUIV="REFRESH" CONTENT=2>
<TITLE>Reload Document</TITLE>
</HEAD>
<BODY>
<P>This document will be reloaded every two seconds.
</BODY>
</HTML>

<HTML>
<HEAD>
<META HTTP-EQUIV="REFRESH" CONTENT="5; URL=http://www.sample.com/next.htm">
<TITLE>Load Next Document</TITLE>
</HEAD>
<BODY>
<P>After five seconds have elapsed, the document
"http://www.sample.com/next.htm" will be loaded.
</BODY>
</HTML>
```

NOBR

<NOBR>

Turns off line breaking. Renders text without line breaks.

Example

```
<NOBR>Here's a line of text I don't want to be broken . . . here's the end
of the line.</NOBR>
```

NOFRAMES

<NOFRAMES>

Indicates content viewable only by browsers that do not support frames. Browsers that support frames will not display content between the beginning and ending NOFRAMES tags. You can create a page that is compatible with both types of browser by using NOFRAMES.

Example

```
<FRAMESET>
 <NOFRAMES>You need Internet Explorer 3.0 to view frames!</NOFRAMES>
</FRAMESET>
```

OBJECT

<OBJECT
ALIGN=LEFT|CENTER|RIGHT
BORDER=*n*
CLASSID=*url*
CODEBASE=*url*
CODETYPE=*codetype*
DATA=*url*
DECLARE
HEIGHT=*n*
HSPACE=*n*
NAME=*url*
SHAPES
STANDBY=*message*
TYPE=*type*
USEMAP=*url*
VSPACE=*n*
WIDTH=*n***>**

Inserts an object, such as an image, document, applet, or control, into the HTML document. The end-tag is required. An object can contain any elements ordinarily used within the body of an HTML document, including section headings, paragraphs, lists, forms, and even nested objects.

ALIGN=LEFT|CENTER|RIGHT

Sets the alignment for the object. The *align-type* can be one of these values:

BASELINE	The bottom of the object aligns with the baseline of surrounding text.
CENTER	The object is centered between left and right margins. Subsequent text starts on the next line after the object.
LEFT	The object aligns with the left margin, and subsequent text wraps along the right side of the object.
MIDDLE	The middle of the object aligns with the baseline of surrounding text.
RIGHT	The object aligns with the right margin, and subsequent text wraps along the left side of the object.
TEXTBOTTOM	The bottom of the object aligns with the bottom of surrounding text.
TEXTMIDDLE	The middle of the object aligns with the midpoint between the baseline and the x-height of the surrounding text.
TEXTTOP	The top of the object aligns with the top of surrounding text.

BORDER=*n*

Specifies the width of the border if the object is defined to be a hyperlink.

CLASSID=*url*

Identifies the object implementation. The syntax of the *url* depends on the object type. For example, for registered ActiveX controls, the syntax is: **CLSID:***class-identifier*.

CODEBASE=*url*

Identifies the code base for the object. The syntax of the *url* depends on the object.

CODETYPE=*codetype*

Specifies the Internet media type for code.

DATA=*url*

Identifies data for the object. The syntax of the *url* depends on the object.

DECLARE

Declares the object without instantiating it. Use this when creating cross-references to the object later in the document or when using the object as a parameter in another object.

HEIGHT=*n*

Specifies the suggested height for the object.

HSPACE=*n*

Specifies the horizontal gutter. This is the extra, empty space between the object and any text or images to the left or right of the object.

NAME=*url*

Sets the name of the object when submitted as part of a form.

SHAPES

Specifies that the object has shaped hyperlinks.

STANDBY=*message*

Sets the message to show while loading the object.

TYPE=*type*

Specifies the Internet media type for data.

USEMAP=*url*

Specifies the image map to use with the object.

VSPACE=*n*

Specifies the vertical gutter. This is the extra, empty space between the object and any text or images above or below the object.

WIDTH=*n*

Specifies the suggested width for the object.

OL

<OL
START=*n*
TYPE=*order-type* >

Draws lines of text as an ordered list. Specifies that the following block consists of individual items, each beginning with an LI tag. The items are numbered.

START=*n*

Specifies a starting number for the list.

TYPE=*order-type*

Changes the style of the list. The *order-type* can be one of these values:

A	Use large letters.
a	Use small letters.
I	Use large Roman numerals.
i	Use small Roman numerals.
1	Use numbers.

Example

```
<OL>
<LI>This is the first item in the list.
<LI>And this is the second item in the list.
</OL>

<OL START=3>
<LI>This is item number 3.
</OL>

<OL TYPE=A>
<LI>This is item A.
</OL>
```

OPTION

<SELECT
SELECTED
VALUE=*value***>**

Denotes one choice in a list box. In a SELECT block, denotes one of the choices that will appear in the list.

SELECTED

Indicates that this item is the default. If not present, item #1 becomes the default.

VALUE=*value*

Indicates the value that will be returned if this item is chosen.

P

<P
ALIGN=LEFT|CENTER|RIGHT>

Inserts a paragraph break and denotes a paragraph.

ALIGN=LEFT|CENTER|RIGHT

Sets the alignment of the paragraph. The *align-type* can be LEFT, CENTER, or RIGHT. Default is left alignment.

The end-tag is optional.

Example

```
<P>This is a paragraph.</P>
```

PARAM

<PARAM
NAME=*name*
VALUE=*value*
VALUETYPE=*type*
TYPE=*type***>**

Sets property values for a given object.

NAME=*name*

Specifies the property name.

VALUE=*value*

Specifies the property value. The value is passed to the object without change except that any character or numeric character entities are replaced with their corresponding character values.

VALUETYPE=*type*

Specifies how to interpret the value. The *type* can be one of these values:

DATA	The value is data. This is the default value type.
REF	The value is a URL.
OBJECT	The value is a URL of an object in the same document.

TYPE=*type*

Specifies the Internet media type.

This element is valid only within an OBJECT element. The end-tag is optional.

PLAINTEXT

<PLAINTEXT>

Renders text in fixed-width type without processing tags. Also turns off HTML parsing until the browser encounters the </PLAINTEXT> tag.

Example

```
<PLAINTEXT> Here's a sample of HTML: <A HREF="sample.url">This is a
shortcut to sample.</A></PLAINTEXT>
```

PRE

<PRE>

Renders text in fixed-width type.

Example

```
<PRE>Here's some plain text.</PRE>
```

Source

HTML 2

S

<S>

Renders text in strikethrough type.

Example

```
<S>This text has a line through it.</S>
```

SAMP

<SAMP>

Specifies sample text. Renders text in a small font. (If no FONT FACE is specified, the font used is fixed-width.)

Example

```
<SAMP>Here is some text in a small fixed-width font.</SAMP>
```

SCRIPT

<SCRIPT
LANGUAGE=*scripting language***>**

Specifies the inclusion of a script. Scripts execute and instantiate objects in the order in which they appear in the HTML. Named objects can be referenced only in the order in which they appear in the document.

LANGUAGE=*scripting language*

Indicates the ActiveX Scripting language in which the enclosed script was written. Examples of an ActiveX Scripting language are "VBScript" and "JScript".

Example

```
<SCRIPT>
<SCRIPT language="VBScript">
    '... Additional VBScript statements ...
</SCRIPT>
```

SELECT

<SELECT
MULTIPLE
NAME=*name*

SIZE=*n*>

Denotes a list box or dropdown list.

MULTIPLE
Indicates that multiple items can be selected.

NAME=*name*
Specifies a name for the list.

SIZE=*n*
Specifies the height of the list control.

Example

```
<SELECT NAME="Cars" MULTIPLE SIZE="1">
    <OPTION VALUE="1">BMW
    <OPTION VALUE="2">PORSCHE
    <OPTION VALUE="3" SELECTED>MERCEDES
</SELECT>
```

SMALL

<SMALL>

Makes text one size smaller.

Example

```
<SMALL>This text is smaller.</SMALL>
```

SPAN

<SPAN>
STYLE=

Applies style information to text within a document. SPAN can be used to do localized formatting to text using STYLE as an attribute. See STYLE

Example

```
<SPAN STYLE="margin-left: 1.0in"> This paragraph is 1.0 inches from the
left margin.<SPAN>
```

STYLE

<STYLE>

Provides a means for including rendering information using a specified style notation. Information in the STYLE element overrides client defaults and that of linked style sheets. It allows authors to specify overrides, while for the most part using a generic style sheet, and as such improves the effectiveness of caching schemes for linked style sheets.

Example

```
<STYLE>
BODY {background: white; color: black}
H1 {font: 14pt Arial bold}
P {font: 10pt Arial; text-indent: 0.5in}
A {text-decoration: none; color: blue}
```

STRIKE

<STRIKE>

Renders text in strikethrough type.

Example

```
<STRIKE>This text has a line through it.</STRIKE>
```

STRONG

<STRONG>

Emphasizes the text. Usually displays the text in bold.

Example

```
<STRONG>This text will be bold.</STRONG>
```

SUB

<SUB>

Renders text in subscript.

Example

```
<SUB>This text is rendered as subscript.</SUB>
```

SUP

<SUP>

Renders text in superscript.

Example

```
<SUP>This text is rendered as superscript.</SUP>
```

TABLE

<TABLE
ALIGN=LEFT|CENTER|RIGHT
BACKGROUND=*url*
BGCOLOR=*color*
BORDER=*n*
BORDERCOLOR=*color*
BORDERCOLORDARK=*color*
BORDERCOLORLIGHT=*color*
CELLPADDING=*n*
CELLSPACING=*n*
COLS=*n*
FRAME=*frame-type*
RULES=*rule-type*
WIDTH=*n*

Creates a table. The table is empty unless you create rows and cells using the TR ,TD and TH elements.

ALIGN=LEFT|CENTER|RIGHT

Specifies the table alignment. The *align-type* can be one of these values:

LEFT	The table is left-aligned. This is the default alignment.
RIGHT	The table is right-aligned. If the table is less than the width of the window, text following the table wraps along the left side of the table.

BACKGROUND=*url*

Specifies a background picture. The picture is tiled behind the text and graphics in the table, table head, or table cell.

BGCOLOR=*color*

Sets background color. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

BORDER=*n*

Sets the size, in pixels, of the table border. The default is zero.

BORDERCOLOR=*color*

Sets border color and must be used with the **BORDER** attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

BORDERCOLORLIGHT=*color*

Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of **BORDERCOLORDARK**, and must be used with the **BORDER** attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

BORDERCOLORDARK=*color*

Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of **BORDERCOLORLIGHT**, and must be used with the **BORDER** attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

CELLPADDING=*n*

Sets the amount of space, in pixels, between the sides of a cell and its contents.

CELLSPACING=*n*

Sets the amount of space, in pixels, between the frame (exterior) of the table and the cells in the table.

COLS=*n*

Sets the number of columns in the table. If given, this attribute may speed up processing of tables, especially lengthy ones.

FRAME=*frame-type*

Specifies which sides of a frame (outer borders) are displayed. The *frame-type* can be one of these values:

VOID	Removes all outside table borders.
ABOVE	Displays a border on the top side of the table frame.
BELOW	Displays a border on the bottom side of the table frame.
HSIDES	Displays a border on the top and bottom sides of the table frame.
LHS	Displays a border on the left-hand side of the table frame.
RHS	Displays a border on the right-hand side of the table frame.
VSIDES	Displays a border on the left and right sides of the table frame.
BOX	Displays a border on all sides of the table frame.
BORDER	Displays a border on all sides of the table frame.

RULES=*rule-type*

Specifies which dividing lines (inner borders) are displayed. The *rule-type* can be one of these values:

NONE	Removes all interior table borders.
GROUPS	Displays horizontal borders between all table groups. Groups are specified by the THEAD , TBODY , TFOOT and COLGROUP elements.
ROWS	Displays horizontal borders between all table rows.
COLS	Displays vertical borders between all table columns.
ALL	Displays a border on all rows and columns.

WIDTH=*n*

Sets the width of the table in pixels or as a percentage of the window. To set a percentage, the *n* must end with a percent (%) sign.

The end-tag is required.

The optional THEAD ,TBODY ,TFOOT ,COLGROUP and COL elements can be used to organize a table and apply attributes across columns and groups of columns.

Example

```
<TABLE BORDER=1 WIDTH=80%>
<THEAD>
<TR>
    <TH>Heading 1</TH>
    <TH>Heading 2</TH>
</TR>
<TBODY>
<TR>
    <TD>Row 1, Column 1 text.</TD>
    <TD>Row 1, Column 2 text.</TD>
</TR>
<TR>
    <TD>Row 2, Column 1 text.</TD>
    <TD>Row 2, Column 2 text.</TD>
</TR>
</TABLE>
```

TBODY

<TBODY>

Defines the table body. Use this element to distinguish the rows in the table header or footer from those in the main body of the table.

If a table does not have a header or footer (does not have a THEAD or TFOOT element), the TBODY element is optional. The end-tag is always optional.

You can use the TBODY element more than once in a table. This is useful for dividing lengthy tables into smaller units and for controlling the placement of horizontal rules.

Example

```
<TABLE>
<THEAD>
<TR>
    ...
</TR>
<TBODY>
<TR>
    ...
</TR>
</TBODY>
</TABLE>
```

TD

<TD
ALIGN=CENTER|LEFT|RIGHT
BACKGROUND=*url*
BGCOLOR=*color*
BORDERCOLOR=*color*
BORDERCOLORLIGHT=*color*
BORDERCOLORDARK=*color*

COLSPAN=*n*
NOWRAP=NOWRAP
ROWSPAN=*n*
VALIGN=*align-type*>

Creates a cell in a table.

ALIGN=CENTER|LEFT|RIGHT
Specifies the horizontal alignment of text in a cell. By default, text is centered.

BACKGROUND=*url*
Specifies a background picture. The picture is tiled behind the text and graphics in the table, table head, or table cell.

BGCOLOR=*color*
Sets background color. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

BORDERCOLOR=*color*
Sets border color and must be used with the **BORDER** attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

BORDERCOLORLIGHT=*color*
Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of **BORDERCOLORDARK**, and must be used with the **BORDER** attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

BORDERCOLORDARK=*color*
Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of **BORDERCOLORLIGHT**, and must be used with the **BORDER** attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

VALIGN=*align-type*
Specifies the vertical alignment of text in the cell. The *align-type* can be one of these values:

TOP	Text is aligned with the top of each cell.
MIDDLE	Text is aligned in the middle of each cell.
BOTTOM	Text is aligned with the bottom of each cell.
BASELINE	Text in adjoining cells in a row is aligned along a common baseline.

By default, text is aligned in the middle of the cell.

This element is valid only within a row in a table, that is, you must use a TR element before using TD. All attributes are optional. The end-tag is optional.

TEXTAREA

<TEXTAREA
COLS=*n*
NAME=*name*
ROWS=*n*>

Creates a multiple-line text entry control in which the user can enter and edit text.

COLS=*n*

Sets the width, in characters, of the text area.

NAME=*n*

Sets the name of the text area. This name is used when the element is used within a FORM element.

ROWS=*n*

Sets the height, in characters, of the text area.

The end-tag is required. Any text between the start-tag and end-tag is used as the initial value for the control.

TFOOT

<TFOOT>

Defines the table footer. Use this element to distinguish the rows in the table footer from those in the header or main body of the table.

The table footer is optional, but if given only one footer is allowed. The TFOOT element is valid only within a table; you must use a TABLE element before using this element. The end-tag is optional.

Example

```
<TABLE>
<TBODY>
     <TR>
     ...
     </TR>
<TFOOT>
     <TR>
     ...
     </TR>
</TABLE>
```

TH

<TH
ALIGN=LEFT|CENTER|RIGHT
BACKGROUND=*url*
BGCOLOR=*color*
BORDERCOLOR=*color*
BORDERCOLORLIGHT=*color*
BORDERCOLORDARK=*color*
COLSPAN=*n*
NOWRAP=NOWRAP
ROWSPAN=*n*
VALIGN=*align-type*>

Creates a row or column heading in a table. The element is similar to the TD element but emphasizes the text in the cell to distinguish it from text in TD cells.

ALIGN=LEFT|CENTER|RIGHT

Specifies the alignment of text in the cell. By default, text is centered.

BACKGROUND=*url*

Specifies a background picture. The picture is tiled behind the text and graphics in the table, table head, or table cell.

BGCOLOR=*color*

Sets background color. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

BORDERCOLOR=*color*

Sets border color and must be used with the **BORDER** attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

BORDERCOLORLIGHT=*color*

Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of **BORDERCOLORDARK**, and must be used with the **BORDER** attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

BORDERCOLORDARK=*color*

Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of **BORDERCOLORLIGHT**, and must be used with the **BORDER** attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

COLSPAN=*n*

Indicates the number of table columns this cell spans.

NOWRAP=NOWRAP

Prevents word wrapping within the cell. Lines of text appear as given in the HTML document.

ROWSPAN=*n*

Indicates the number of table rows this cell spans.

VALIGN=*align-type*

Specifies the vertical alignment of text in the table. The *align-type* can be one of these values:

TOP	Text is aligned with the top of each cell.
MIDDLE	Text is aligned in the middle of each cell.
BOTTOM	Text is aligned with the bottom of each cell.
BASELINE	Text in adjoining cells in a row is aligned along a common baseline.

By default, text is aligned in the middle of the cell.

This element is valid only within a row in a table, that is, you must use a TR element before using TH. All attributes are optional. The end-tag is optional.

THEAD

<THEAD>

Defines the table header. Use this element to distinguish the rows in the table header from those in the footer or main body of the table.

The table header is optional, but if given only one header is allowed. The THEAD element is valid only within a table; you must use a TABLE element before using this element. The end-tag is optional.

Example

```
<TABLE>
<THEAD>
    <TR>
    ...
    </TR>
<TBODY>
    <TR>
    ...
    </TR>
</TABLE>
```

TITLE

<TITLE>

Specifies a title for the document. Internet Explorer uses this for the window caption.

This element is valid only within the HEAD element. The end-tag is required.

Example

```
<HEAD>
<TITLE>"Welcome To Internet Explorer!"</TITLE>
</HEAD>
```

TR

<TR
ALIGN=CENTER|LEFT|RIGHT
BACKGROUND=*url*
BGCOLOR=*color*
BORDERCOLOR=*color*
BORDERCOLORLIGHT=*color*
BORDERCOLORDARK=*color*
VALIGN=*align-type***>**

Creates a row in a table.

ALIGN=CENTERLEFT|RIGHT

BACKGROUND=*url*
Specifies a background picture. The picture is tiled behind the text and graphics in the table, table head, or table cell.

BGCOLOR=*color*
Sets background color. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

BORDERCOLOR=*color*

Sets border color and must be used with the **BORDER** attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

BORDERCOLORLIGHT=*color*

Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of **BORDERCOLORDARK**, and must be used with the **BORDER** attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

BORDERCOLORDARK=*color*

Sets independent border color control over one of the two colors used to draw a 3-D border, opposite of **BORDERCOLORLIGHT**, and must be used with the **BORDER** attribute. The *color* is either a hexadecimal, red-green-blue color value or a predefined color name. See Color

VALIGN=*align-type*

Specifies the vertical alignment of text in the cells of the row. The *align-type* can be one of these values:

TOP	Text is aligned with the top of each cell.
MIDDLE	Text is aligned in the middle of each cell.
BOTTOM	Text is aligned with the bottom of each cell.
BASELINE	Text in adjoining cells in a row is aligned along a common baseline.

By default, text is aligned in the middle of the cell.

TT

<TT>

Indicates teletype. Renders text in fixed-width type.

Example

```
<TT>Here's some plain text.</TT>
```

U

<U>

Renders text underlined.

Example

```
<U>This text has a line under it.</U>
```

UL

<UL>

Draws lines of text as a bulleted list. Specifies that the following block consists of individual items, each beginning with an LI tag. The items are bulleted.

Example

```
<UL>
<LI>This is the first bulleted item in the list.
<LI>And this is the second bulleted item in the list.
</UL>
```

VAR

<VAR>

Indicates placeholder text for a variable. Displays text in a small, fixed-width type.

Example

```
Enter the <VAR>filename</VAR> in the dialog box.
```

WBR

Inserts a soft line break in a block of NOBR text.

Example

```
<NOBR> This line of text will not break, no matter how narrow the window
gets.<WBR> This one, however<WBR>, will.</NOBR>
```

XMP

<XMP>

Indicates example text. Displays text in fixed-width type.

Example

```
<XMP>Here's some plain text.</XMP>
```

APPENDIX R
OLEDOC~1.DOC

OLE Document Objects Specifications

Version 1.0, Also Known as "DocObjects"

Distribution: Public

OLE Design Team and Office Design Team
27 February, 1996

Note: This document is an early release of the final specification. It is meant to specify and accompany software that is still in development. Some of the information in this documentation may be inaccurate or may not be an accurate representation of the functionality of the final specification or software. Microsoft assumes no responsibility for any damages that might occur either directly or indirectly from these inaccuracies. Microsoft may have trademarks, copyrights, patents or pending patent applications, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give you a license to these trademarks, copyrights, patents, or other intellectual property rights.

Contents

Introduction

The software industry has entered an era in which customers rely on many products to complete their work. For example, when a new company creates its business plan, it may rely on Microsoft Word to create the basic proposal, Excel to create a summary of projected financial performance, and PowerPoint to create a slide show for potential investors. Although customers rely on several distinct products to complete their projects, they think of each project as a single entity: a business plan, a sales proposal, a book, and so forth.

Office for Windows 95 introduced a new application called the Binder, that makes it easier for customers to complete projects that contain heterogeneous documents (that is, a variety of documents created by many distinct applications). And it makes it easy for them to use the standard Office applications as they do so. It may be helpful to think of the Binder as a an electronic paper clip: it holds together text files, spreadsheets, graphics presentations and other documents so that the user can manipulate them as a single entity. From another perspective, the Binder is a sophisticated "viewer" that can host a variety of heterogeneous documents which let the user create, edit, save, print, and view distinctly different kinds of information.

"Document Objects" (DocObjects for short) is the core technology that makes Office Binder work. While originally developed as a proprietary technology for Microsoft Office, Microsoft believe that DocObjects represents an important step forward and that is will benefit customers in many more ways than just Office and Office-Compatible applications. Most notably is that the technology is flexible enough to support document containers other than Office Binder, and can support document servers other than just Office and Office Compatible applications.

One obvious application for this technology is within the domain of "Internet browsers", where the adoption of Binder technology will not only facilitate the presentation of Internet-based information (Web pages and so forth) but will, at the same time with the same implementation, enable the browser to present documents from Office and Office Compatible applications. In short, the user need only go to one navigation tool to browse and view all documents whether local or network-based.

This specification explains the Document Objects architecture in terms of the container and the server side of the technology.

Note: This specification has been written from the perspective of an advanced OLE programmer. For additional information about OLE issues discussed below, please consult the OLE Programmer's Reference and related publications.

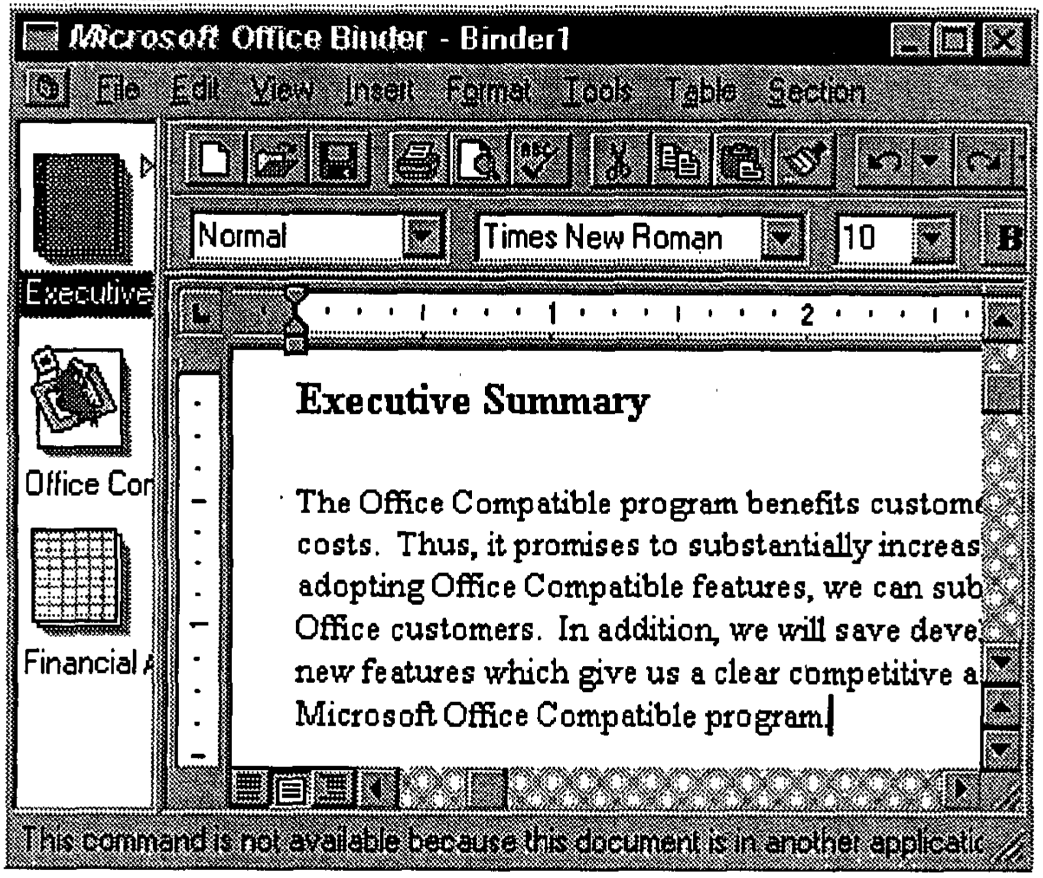
The current Internet Explorer does not implement all features described in this specification. It's features are as follows:

1. Internet Explorer only supports one document view as a container. It also views only one document at a time (per instance of IE).
2. Internet Explorer makes no use of IPrint.
3. Internet Explorer does route a number of commands through IOleCommandTarget.

This document does not discuss the means for doing content-indexing on the contents of a non-HTML document as would be involved with files viewed using Document Objects. See the specification for the IFilter interface for more information.

Feature Description

The following picture illustrates the Office 95 Binder, which for the purposes of this document serves merely as an illustration of a "document container."



Executive Summary

The Office Compatible program benefits custom costs. Thus, it promises to substantially increas adopting Office Compatible features, we can sub Office customers. In addition, we will save deve new features which give us a clear competitive a Microsoft Office Compatible program.

This command is not available because this document is in another applicatic

As this illustration shows, the Binder includes two primary panes, as will most containers. The left pane shows thumbnails that correspond to the section of the Binder. For example, the preceding Binder contains a Word document, a PowerPoint slide show, and an Excel spreadsheet. Users can click these images to activate the corresponding "document object" or "DocObject." The right pane of the Binder shows the document on which the user is currently working (a Word document in the preceding illustration). In an "Explorer" type of container, the left pane would display a hierarchical tree list of a drive or network while the right pane would display the available document or page in that point of the hierarchy.

When a document is "activated" in the right hand pane, it looks and acts, for all intents and purposes, as if the user was running the stand-alone application that normally manages that particular document type, complete with toolbars, menus, and all other user interface elements. A container like Office Binder thus provides a single frame in which to work with documents, instead of forcing the user to multiple application frames for each document type. (This is also different than working with embeddings in a compound document where only a single piece of content is being activated; here we are activating an entire document, that is, an entire application, within the context of a single frame).

While the Document Objects technology allows an application like Office Binder to present literally a "three-ring binder" paradigm, DocObjects is generic enough to accommodate many other possible user interfaces that have the same requirements.

A General Overview of Document Objects

The DocObjects technology is a set of extensions to OLE Documents, the compound document technology of OLE. The extensions are in the form of additional interfaces that allow what mostly looks like an embeddable in-place object to represent an entire document instead of a single piece of embedded content. As with OLE Documents, DocObjects involve a container that provides the display space for DocObjects and servers that provide the user interface and manipulation capabilities for DocObjects themselves.

A DocObject server is a product that supports one or more document object classes, where each object itself supports the extension interfaces that allow the object to be activated in a suitable container, such as Binder. A DocObject is best understood by distinguishing it from a standard OLE embedded object. Following the OLE convention, an embedded object is one which is displayed within the "page" of the document that "owns it" where the document is managed by an OLE container. The container stores the embedded object's data with the rest of the document.

However, embedded objects are limited in that they do not control the page on which they appear. By necessity they tend to be rather small objects: a picture that supplements the surrounding text (provided by the container), a spreadsheet that clarifies its supporting analysis (again provided by the container), and so forth.

By contrast, a document object provided from a DocObject server is essentially a full-scale, conventional document which is embedded as an object within another DocObject container (Binder, browsers, etc.). Unlike embedded objects, DocObjects have complete control over their pages, and the full power of the application is available to the user to edit them. Thus, unlike embedded objects, DocObjects tend to be full-scale, robust documents that exploit the complete native functionality of the server (application) that creates them. Users can create documents (called sections within the Binder, for example) using the full horsepower of their favorite applications (if they are DocObject enabled), yet they can treat the resulting project as a single entity, which can be uniquely named, saved, transmitted to coworkers for review or editing, printed as a single entity, and so forth. In the same way, a user of an Internet browser (such as a future Explorer) can treat the entire network as well as local file systems as a single document-storage entity with the ability to browse the documents in that storage from a single location.

Summary of Requirements for Document Object Participation

A DocObject container that wishes to integrate DocObjects must:

1. Be capable of handling object storage through the *IPersistStorage* interface, that is, it must provide an *IStorage* instance to each DocObject.
2. Support the basic embedding features of OLE Documents, necessitating "site" objects (one per document or embedding) that implements *IOleClientSite* and *IAdviseSink.*
3. Support in-place activation of embedded objects or DocObjects, requiring the container's site objects to implement *IOleInPlaceSite* and requiring the container's frame object to provide *IOleInPlaceFrame.*
4. Support the DocObjects extensions through the implementation of *IOleDocumentSite* and possibly IContinueCallback on the site object, along with *IOleCommandTarget* on the frame object.

(Note that OLE Documents support in a container implies more than just interface implementations: it also requires knowledge of using the interfaces of an embedded object. Same applies to DocObjects extensions where the container must also know how to use those extension interfaces on the DocObjects themselves.)

Correspondingly, a "document object" that wishes to work within a DocObject container must:

1. Use OLE's Compound Files as their storage mechanism, that is, implement *IPersistStorage.*
2. Support the basic embedding features of OLE Documents, including "Create From File." This necessitates the interfaces *IPersistFile*, *IOleObject*, and *IDataObject.*
3. Support the in-place activation extension of OLE Documents, that is, *IOleInPlaceObject* and *IOleInPlaceActiveObject* (using the container's *IOleInPlaceSite* and *IOleInPlaceFrame* interfaces).
4. Support the DocObjects extensions that involves these new interfaces: *IOleDocument*, *IOleDocumentView*, *IOleCommandTarget*, and *IPrint.*

Again, knowledge of when and how to use the container-side interfaces is implied in these requirements.

The remainder of this document will describe the architecture of Document Objects and how the new interfaces of *IOleDocument*, *IEnumOleDocumentViews*, *IOleDocumentView*, *IOleDocumentSite*, *IPrint*, *IContinueCallback*, and *IOleCommandTarget* work together to achieve the document-activation features described earlier. First will be the architectural details of Document Objects including special requirements for Help menu merging (an extension to OLE Documents), programmatic printing, and "command targets." This is followed by implementation notes for DocObject containers and servers, followed by the complete interface reference and an appendix describing additional details of the Office Binder's implementation this architecture.

Architectural Details of Document Objects

Overview

In general, a particular product owns a set of data, the storage in which they are saved, and the views through which they are displayed for the user. OLE Documents introduced technology that let applications store their documents (including data and information about the way they should be displayed) in an abstract manner. That is, the application was freed from the need to understand the granular aspects of its storage vehicle, and it could, instead, deal with a variety of storage sites in a consistent fashion (that is, without regard for their underlying properties). Products that exploited this OLE "abstract storage technology", could save their documents into files, databases, and other types of storage in a uniform way. In addition, OLE enabled applications (via *IPersist** interfaces) to permanently save their documents in storage that they did not own. Thus, applications that support OLE can treat storage abstractly.

The OLE Documents architecture also took an important step toward letting applications treat their views (that is, the port through which their data are displayed for the user) abstractly. Products that supported in-place activation could display their content in a "foreign" frame that they did not own. While this represented an important step forward, it had its limitations. In particular, in-place activation supports an object view of the data rather than a document view. That is, the OLE container is responsible for siting the object's view port, for controlling the overall display of its pages, for printing them, and so forth. To overcome some of these limitations, the embedded object can be opened so that more of the native functionality of its parent (that is, the OLE server) can be used. Nevertheless, some important limitations remained.

Among these limitations, it is worth noting that a defining characteristic of a document, as distinguished from an embedded object, is that it owns the printed page. It can have headers, footers, footnotes, endnotes, revision marks, and so forth, and it knows where to place them on the page and how to display them for the user. Embedded objects (that is, object views) do not control the page on which they appear. Instead, they must live within a containment hierarchy whose root container is an actual document.

The Document Objects architecture defines an abstraction for *views* and their management, so that objects can function within containers and yet retain control over their display and some important printing functions. This architecture makes it possible to display documents both in foreign frames (such as Binder or Explorer) and in native frames (such as the product's own view ports).

A view can be divided into two components: the view frame component and the view component. The view frame could consist of just the frame window in the case of an Single Document Interface (SDI) product, or it could include the frame and the Multiple Document Interface (MDI) window in the case of an MDI product. The view frame component provides the space for menus, toolbars, a status bar, and so forth. The frame component also provides the view port within which adornments such as rulers, scroll bars, and similar tools can be displayed. Note that the frame does not "tell the document" how big it should be, nor does it care. On the contrary, it merely conveys to the contained application the view port size that was selected by the user. The view port, by contrast, is the region within which the data themselves are displayed. If the frame were an SDI frame, for example, then the view port could be the client area of the frame window minus the space allocated for tool bars, status bar, and such. In an MDI setting, the view port would be the client area of the MDI document window minus any other frame level user interface elements (for example, space for tab bar in case of workbook).

This breakdown of components offers several advantages:

- The view frame can be of any type: SDI, MDI, Workbook, Form, and so forth. A single implementation of the document's view can be used within many different types of frames.

- Multiple applications can have the same frame level UI and functionality, while supporting distinctly different data sets. In principle, this offers a great advantage to vendors who wish to develop a core user interface as a frame that is used throughout their entire product line. In essence, they can build the frame once and re-use it as appropriate.

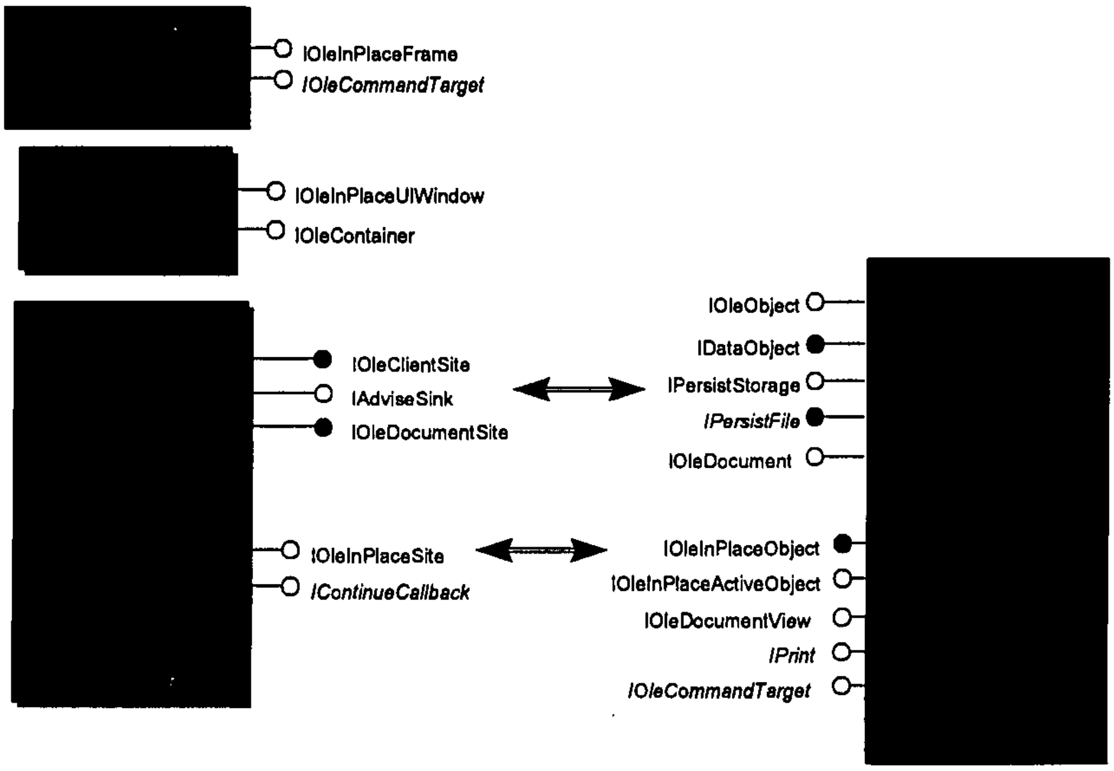
Special attention should be given to the fact that the view and storage aspects of a data set are two entirely orthogonal aspects of the document to which they belong. The storage provider and the view frame provider could be the same, or they could be different. In any case, the application can proceed with its work in a standard manner that isolates it from the need to understand either its storage or its view port in specific detail. In some sense, this separation of views and storage is present within OLE, since when an embedded object is "open edited", the server application provides the view frame while its container provides the storage. However, the Document Objects architecture takes matters much further, since it lets the document's storage container (or some other container) provide the view frame. In addition, moniker binders can also provide view frames, and this can enable in-place activation of links.

Relevant Objects and the Interfaces They Must Implement

A DocObject can support one or more views, each of which is capable of in-place activation. The document component of the object must support standard OLE Document interfaces, but in addition it must support new interfaces such as *IOleDocument*. The view component must also support certain standard OLE interfaces (*IOleInPlaceObject* and *IOleInPlaceActiveObject*), and in addition it must support the new *IOleDocumentView* interface, too. DocObject containers must implement *IOleDocumentSite* along with OLE container interfaces, and they must implement *IOleInPlaceSite* on each view site. Finally, the frame object, the view object(s), and the container object can optionally implement *IOleCommandTarget* to support the dispatch of certain commands (as discussed below).

View and container objects can also optionally implement *IPrint* and *IContinueCallback* (discussed below), to support programmatic printing.

The following illustration shows the conceptual relationships between a container and its components (on the left) and the DocObject and its views (on the right), where the DocObject manages storage and data and the view displays and possibly prints that data. Interfaces in bold are those required for DocObject participation; those bold and italic are optional. All others are required according to OLE Document rules:



Note that a document that supports only a single view can implement both the view and document components (that is, their corresponding interfaces) on a single concrete class. In addition, a container site that only supports one view at a time can combine the document site and the view site into a single concrete site class. The container's frame object, however, must remain distinct, and the container's document component is merely shown here to complete the OLE Documents architecture. This piece of the container is not affected by the Document Objects architecture.

A DocObject is one that has some data and one or more views associated with it. The Document Objects architecture formalizes the relationship between the document, its views, and their view sites/frames.

Document Objects (Server)

The DocObject owns a set of data and has access to storage where the data can be saved and retrieved. It can create and manage one or more views on its data. In addition to supporting the usual embedding and in-place activation interfaces according to OLE Documents, the DocObject communicates its ability to create views through *IOleDocument*. Through this interface the container can ask to create (and possibly enumerate) the views that the DocObject can display. Through this interface, the DocObject can also provide miscellaneous information about itself, such as whether it supports multiple views or complex rectangles.

```
interface IOleDocument : IUnknown
    {
    HRESULT CreateView([in] IOleInPlaceSite *pIPSite, [in] IStream *pstm
        , [in] DWORD dwReserved, [out] IOleDocumentView **ppView);
    HRESULT GetDocMiscStatus([out] DWORD *pdwStatus);
    HRESULT EnumViews([out] IEnumOleDocumentViews **ppEnum
        , [out] IOleDocumentView **ppView);
    }
```

Every DocObject must have a view frame provider with this interface. If the document is not embedded within a container, then the DocObject server itself must provide the view frame. However, when the DocObject is embedded in a DocObject container then the container provides the view frame.

The *IEnumOleDocumentViews* interface is a standard OLE enumerator for *IOleDocumentView* * types.

Views Objects (Server)

A DocObject can create one or more types of views (for example, normal, outline, page layout, etc.) of its data. From a functional perspective, views act like filters through which the data can be seen.

Even if the document has only one type of view, it may still wish to support multiple views as a means of supporting "Window/New Window" functionality (for example, the Window menu in Office applications). Functionally these views are like ports onto a particular method for displaying the data.

To be represented within the a DocObject container, a view component must support *IOleInPlaceObject* and *IOleInPlaceActiveObject* in addition to *IOleDocumentView*:

```
interface IOleDocumentView : IUnknown
    {
    HRESULT SetInPlaceSite([in] IOleInPlaceSite *pIPSite);
    HRESULT GetInPlaceSite([out] IOleInPlaceSite **ppIPSite);
    HRESULT GetDocument([out] IUnknown **ppunk);
    [input_sync] HRESULT SetRect([in] LPRECT prcView);
    HRESULT GetRect([in] LPRECT prcView);
    [input_sync] HRESULT SetRectComplex([in] LPRECT prcView
        , [in] LPRECT prcHScroll, [in] LPRECT prcVScroll
        , [in] LPRECT prcSizeBox);
    HRESULT Show([in] BOOL fShow);
    HRESULT UIActivate([in] BOOL fUIActivate);
    HRESULT Open(void);
    HRESULT CloseView([in] DWORD dwReserved);
    HRESULT SaveViewState([in] IStream *pstm);
    HRESULT ApplyViewState([in] IStream *pstm);
    HRESULT Clone([in] IOleInPlaceSite *pIPSiteNew, [out] IOleDocumentView
**ppViewNew);
    }
```

Every view has an associated view site, which encapsulates the view frame and the view port (HWND and a rectangular area in that window). The site exposes this functionality though the standard *IOleInPlaceSite* interface. Note that it is possible to have more than one view port on a single HWND.

Typically each type of view has a different printed representation. Hence views and the corresponding view sites should implement the printing interfaces if *IPrint* and *IContinueCallback*, respectively. The view frame must negotiate with the view provider through *IPrint* when printing begins, so that headers, footers, margins, and related elements are printed correctly. The view provider notifies the frame of printing-related events through *IContinueCallback*. For more information on the use of these interfaces, see "Programmatic Printing" later in this document.

Do note that if a DocObject only supports a single view, then the DocObject and that single view can be implemented using a single concrete class. *IOleDocument::CreateView* simply returns the same object's *IOleDocumentView* interface pointer. In short, it is not necessary that there be two separate object instances when only one view is required.

A view object can also choose to be a command target by implement *IOleCommandTarget*. This allows it to easily receive commands that originate in the container's user interface (such as File New, Open, SaveAs, Print; Edit Copy, Paste, Undo, etc.). For more information, see "Command Targets" later in this document.

Document Site Objects (Container)

In the Document Objects architecture, a document site is the same as a client site object in OLE Documents with the addition of the *IOleDocument* interface:

```
interface IOleDocumentSite : IUnknown
    {
    HRESULT ActivateMe(IOleDocumentView *pViewToActivate);
    }
```

The document site is conceptually the container for one or more "view site" objects that are each associated with individual view objects of the document managed by the document site. If the container only supports a single view per document site, then it can implement the document site and the view site with a single concrete class.

View Site Objects (Container)

A container's "view site" object manages the display space for a particular view of a document. In addition to supporting the standard *IOleInPlaceSite* interface, a view site also generally implements *IContinueCallback* for programmatic printing control. (Note that the view object never queries for *IContinueCallback* so it can actually be implemented on any object the container desires).

A container that supports multiple views must be able to create multiple view site objects within the document site. This provides each view with separate activation and deactivation services as provided through *IOleInPlaceSite.*

Frame Object (Container)

The container's frame object is, for the most part, the same frame that is used for in-place activation in OLE Documents, that is, the one that handles menu and toolbar negotiation. A view object has access to this frame object through *IOleInPlaceSite::GetWindowContext* which also provides access to the container object representing the container document (which can handle pane-level toolbar negotiation and contained object enumeration).

In Document Objects, a container can augment the frame by adding *IOleCommandTarget*. This allows it to receive commands that originate in the DocObject's user interface in the same way that this interface can allow a container to send the same commands (such as File New, Open, SaveAs, Print; Edit Copy, Paste, Undo, etc.) to a DocObject. For more information, see "Command Targets" later in this document.

Help Menu Merging: An Extension to OLE Documents

When an object is active within a container, the menu merging protocol of OLE Documents gives the object complete control of the Help menu. As a result, the container's Help topics are not available unless the user deactivates the object. The Document Objects architecture expands on the rules for in-place menu merging to allow both the container and an active DocObject to share the menu. The new rules are simply additional conventions about what component owns what part of the menu and how the shared menu is constructed.

The new convention is simple. In DocObjects, the Help menu has two top-level menu items organized as follows:

```
Help
    Container Help >
    Object Help    >
```

For example, when a Word section is active in the Office Binder, then the Help menu would appear as follows:

```
Help
    Binder Help >
    Word Help   >
```

Both menu items are cascade menus under which any additional menu items specific to the container and the object are provided to the user. What items appear here will vary with the container and objects involved.

To construct this merged Help menu, the Document Objects architecture modifies the normal OLE Documents procedure. According to OLE Documents, the merged menu bar can have 6 groups of menus, namely *File, Edit, Container, Object, Window, Help*, in that order, and in each group there can be 0 or more menus. The groups *File, Container* and *Window* belong to the container and the groups *Edit, Object* and *Help* belong to the object. When the object wants to do menu merging it creates a blank menu bar and hands it over to the container, to let it insert its menus, by calling *IOleInPlaceFrame::InsertMenus*. The object also hands over a structure which is an array of six LONGs (OLEMENUGROUPWIDTHS). After inserting the menus, container would mark how many menus he added in each one of its groups, and then returns. Then the object inserts its menus paying attention to the count of menus in each container group. Then finally object passes the merged menu bar and the array (which contains the count of menus in each group) to OLE, which returns an opaque "menu descriptor" handle. Later the object passes that handle and the merged menu bar to the container, via *IOleInPlaceFrame::SetMenu*. At this time container displays the merged menu bar and also passes the handle to OLE, so that OLE can do proper dispatching of menu messages.

In the modified DocObject procedure, the object must first initialize the OLEMENUGROUPWIDTHS elements to zero before passing it to the container. Then the container would do what it normally does in its menu insertion code with one exception. The container inserts a **Help** pop-up menu as the last item and stores a value of 1 in the last (sixth) entry of the OLEMENUGROUPWIDTHS array (that is, *width[5]* which belongs to the object's Help group). This **Help** popup menu will have only one item which is another popup menu, the "Container Help >" cascade menu as described above.

The object then does its normal menu insertion code, except that before inserting its help menu, it checks the sixth entry of the OLEMENUGROUPWIDTHS array. If the value is 1 and the name of the last menu is **Help** *(or the appropriate localized string)*, then the object inserts its help popup menu as sub-menu of container's **Help** popup menu.

The object then sets the sixth element of OLEMENUGROUPWIDTHS to zero and increments the fifth element by one. This lets OLE know that the **Help** menu belongs to the container and the menu messages corresponding that menu (and its sub menus) should be routed to the container. It is then the container's responsibility to forward WM_INITMENUPOPUP, WM_SELECT, WM_COMMAND, and other menu-related messages that belong to the object's portion of the help menu.[1] The container should use the window returned from the object's *IOleInPlaceActiveObejct::GetWindow* function as the destination for these messages.

If the object detects a zero in the sixth element of OLEMENUGROUPWIDTHS it otherwise proceeds according to the normal OLE Documents rules. This will take care of containers that do participate in help menu merging as well as those which do not.

When the object calls *IOleInPlaceFrame::SetMenu*, before displaying the merged menu bar, the container checks whether his **Help** popup menu has an additional sub-menu, in addition to what it has inserted. If so the container would leave his **Help** popup menu in the merged menu bar, else he will remove it from the merged menu bar. This will take care of the objects that do participate in help menu merging as well as those that do not.

[1] This is accomplished by using WM_INITMENU to clear a flag that tells the container whether or not the user has navigated into the object's Help menu. The container then watches WM_MENUSELECT for entry into or exit from any item on the Help popup that the container did not add itself. On entry, it means the user has navigated into an object popup, so the container sets the "in object Help menu" flag and uses the state of that flag to forward any WM_MENUSELECT, WM_INITMENUPOPUP, and WM_COMMAND messages, as a minimum, to the object window. On exit, the container clears the flag and then processes these same messages itself.

Finally, during menu disassembling time, the object would remove the inserted help menu, in addition to removing the other inserted menus. And when container gets a chance to removed its menus, it will remove its help popup menu in addition to the other menus that it has inserted.

Programmatic Printing (IPrint & IContinueCallback)

OLE provided the means to uniquely identify persistent documents (*GetClassFile*) and load them into their associated code (*CoCreateInstance, QueryInterface(IID_IPersistFile/IID_IPersistStorage...), IPersistFile/IPersistStorage::Load)*. To further enable printing of documents, Document Objects (using an existing OLE design not shipped with OLE 2.0 originally) introduces a base-standard printing interface, *IPrint*, generally available through any object which can load the persistent state of the document type. Each view of a document object in the Document Objects architecture can optionally support the *IPrint* interface to provide these capabilities.

The *IPrint* interface is defined as follows:

```
interface IPrint : IUnknown
    {
    HRESULT SetInitialPageNum([in] LONG nFirstPage);
    HRESULT GetPageInfo([out] LONG *nFirstPage, [out] LONG *pcPages);
    HRESULT Print([in] DWORD grfFlags, [in,out] DVTARGETDEVICE **pptd
        , [in,out] PAGESET ** ppPageSet , [in,out] STGMEDIUM **ppstgmOptions
        , [in] IContinueCallback* pCallback, [in] LONG nFirstPage
        ,[out] LONG *pcPagesPrinted, [out] LONG *pnPageLast);
    };
```

Clients and containers simply use *IPrint::Print* to instruct the document to print itself once that document is loaded, specifying printing control flags, the target device, the pages to print, and additional options. The client can also control the continuation of printing through the interface *IContinueCallback* (see below).

In addition, *IPrint::SetInitialPageNum* supports the ability to print a series of documents together as if they were one by numbering pages seamlessly, obviously a benefit for DocObject containers like Office Binder. *IPrint::GetPageInfo* simply allows the caller to retrieve the starting page number previously passed to *SetInitialPageNum* (or the document's internal default starting page number) and the number of pages in the document, useful for displaying pagination information.

Objects that support *IPrint* mark themselves in the registry with the "Printable" key stored under the object's CLSID:

```
HKEY_CLASSES_ROOT\CLSID\{...}\Printable
```

IPrint is usually implemented on the same object supporting either *IPersistFile* or *IPersistStorage*. Callers note the capability to programmatically print the persistent state of some class by looking in the registry for the "Printable" key. At the time being, "Printable" indicates support for at least *IPrint*; other interfaces may be defined in the future which would then be available through *QueryInterface* where *IPrint* simply represents the base level of support.

During a print procedure, the client or container that initiated the printing may wish to control whether or not the printing should continue. For example, the container may support a "Stop Print" command that should terminate the print job as soon as possible. To support this capability, the client of a printable object can implement a small notification sink object with the interface *IContinueCallback:*

```
interface IContinueCallback : IUnknown
    {
    HRESULT FContinue(void);
    HRESULT FContinuePrinting([in] LONG cPagesPrinted, [in] LONG nCurrentPage
        , [in] LPOLESTR pszPrintStatus);
    };
```

This interface is designed to be useful as a generic continuation callback function which takes the place of the various continuation procedures in the Win32 API (such as the *AbortProc* for printing and the *EnumMetafileProc* for metafile enumeration). Thus this interface design is useful in a wide variety of time-consuming processes.

In the most generic cases, *IContinueCallback::FContinue* function is called periodically by any lengthy process. The sink object returns S_OK to continue the operation, S_FALSE to stop the procedure as soon as possible.

FContinue, however, is not used in the context of *IPrint::Print*; rather, printing uses *IContinueCallback::FContinuePrint*. Any printing object should periodically call *FContinuePrinting* passing the number of pages that have been printing, the number of the page being printed, and an additional string describing the print status that the client may choose to display to the user (such as "Page 5 of 19").

Complete details of these interfaces is given in the reference section at the end of this document.

Command Targets

The command dispatch interface *IOleCommandTarget* defines a simple and extensible mechanism to query and execute commands. This mechanism is simpler than OLE Automation's *IDispatch* because it relies entirely on a standard set of commands, commands rarely have arguments, and no type information is involved (type safety is diminished for command arguments as well).

In this design, each command belongs to a "command group" which is itself identified with a GUID. Therefore anyone can define a new group and define all the commands within that group without any need to coordinate with Microsoft nor any other vendor.[2]

IOleCommandTarget handles the following scenarios:

1. When an object is in-place activated, only the object's toolbars are typically displayed and the object's toolbars may have buttons for some of the container commands like "Print," "Print Preview," "Save," "New," "Zoom," etc.[3] Currently there is no mechanism for the object to dispatch these commands to the container.
2. When a DocObject is embedded in a DocObject container (such as Binder), the container may need to send commands such "Print," "Page Setup," "Properties," etc. to the contained DocObject.

Obviously this simple command routing could be handled through existing OLE Automation standards and *IDispatch*. However, the overhead involved with *IDispatch* is more than is necessary here, so *IOleCommandTarget* provides a simpler means to achieve the same ends:

```
interface IOleCommandTarget : IUnknown
    {
    HRESULT QueryStatus([in] GUID *pguidCmdGroup, [in] ULONG cCmds
        , [in,out][size_is(cCmds)] OLECMD *prgCmds, [in,out] OLECMDTEXT
*pCmdText);
    HRESULT Exec([in] GUID *pguidCmdGroup, [in] DWORD nCmdID, [in] DWORD
nCmdExecOpt
        , [in] VARIANTARG *pvaIn, [in,out] VARIANTARG *pvaOut);
    }
```

The *QueryStatus* method here tests whether a particular set of commands, the set being identified with a GUID, is supported. This call fills an array of OLECMD values (structures) with the supported list of commands as well as returning text describing the name of a command and/or status information. When the caller wishes to invoke a command, it can pass the command (and the set GUID) to *Exec* along with options and arguments, getting back a return value.

For more information on this interface, see the reference section at the end of this document.

[2] This is essentially the same means of definition as a dispinterface plus dispIDs in OLE Automation. There is overlap here, although this command routing mechanism is just for command routing and not for scripting/programmability on a large scale as OLE Automation handles.

[3] In-place activation standards recommend that objects remove such buttons from their toolbars, or at least disable them. This design allows those commands to be enabled and yet routed to the right handler.

Implementation Notes

Becoming a DocObject Server

This section discusses issues related to server side implementation of the Document Objects architecture, specifically the implementation of a DocObject and its view.

A DocObject can be implemented as an in-process object or as a local object (in an EXE). The Document Objects architecture has been designed so that it is relatively easy to transform an existing in-place implementation into a DocObject. The document object itself must support those interfaces described earlier which will require existing object implementations to slightly modify their code in several places: *IOleObject::DoVerb*, *IOleObject::SetClientSite*, and in-place activation functions. The following sections describe these issues in more detail.

IOleObject::SetClientSite

An object must be able to determine whether it can and should activate as a DocObject. This will depend on whether the client site (that is, the container) supports *IOleDocumentSite*. When an object's *IOleObject::SetClientSite* is called, it should query the given pointer for *IOleDocumentSite* as the following code illustrates:

```
HRESULT IOleObject::SetClientSite(IOleClientSite *pSite)
    {
    //Perform regular SetClientSite processing.

    // If we currently have a document site pointer, release it.
    if (NULL!=m_pDocSite)
        {
        ReleaseInterface(m_pDocSite);  //Macro to Release and NULL
        m_fDocObj=FALSE;
        }

    if (NULL!=pSite)
        {
        if (SUCCEEDED(pSite->QueryInterface(IID_IOleDocumentSite,
&m_pDocSite)))
            m_fDocObj=TRUE;
        }
    }
```

IOleObject::DoVerb

When a DocObject's *IOleObject::DoVerb* is called, it will know whether to activate itself as a DocObject or not as determined in *IOleObject::SetClientSite*. One DocObject support is acknowledged, various verbs are handled differently than a normal embedded object would handle them.

Verb	Handling Procedure
OLEIVERB_SHOW	The object calls *IOleDocumentSite::ActivateMe*. The object does not call *IOleClientSite::ShowObject* nor *IOleClientSite::OnShowWindow* at this time because it waits until calls to *IOleDocumentView* for specific activation instructions.
OLEIVERB_OPEN	Same as OLEIVERB_SHOW—note that this is *not recommended* for containers.
OLEIVERB_UIACTIVATE	Same as OLEIVERB_SHOW.
OLEIVERB_HIDE	The object should return an error (E_INVALIDARG)

In-Place Activation Differences

When activating as a DocObject, the object should behave as follows:

- Bypass displaying the in-place hatch border and object adornments (such as sizing handles etc.)

- Do not generate *IOleInPlaceSite::OnPosRectChange* calls (no need for them)
- Ignore *IOleObject::SetExtent* calls
- Draw scroll bars within the view rectangle (see *IOleDocumentView::SetRect* and *SetRectComplex*) as opposed to drawing them outside that rectangle (as in normal in-place activation)
- Do not call *IOleClientSite::ShowObject* during activation.

Storage requirements.

The storage format of a DocObject must be the same whether it opens the file on its own and writes the data or whether it saves that data into storage provided by its container. In short, the DocObject must depends on *IStorage* and *IStream* for its persistence mechanisms. This enables a DocObject container to take the data in the object's storage and create a file out of it. Binder, for example, uses this mechanism to move the bound sections on to the shell.

In standard OLE, when *IPersistFile::Save* method is called with NULL for the file name, then the object must save itself into the file that it currently owns. The frame provider, which is not the storage provider, can use this mechanism to ask the document to save itself into the storage it currently owns.

Registration

Every DocObject server should include the "DocObject" key in the registry entries of its supported classes. This key indicates Document Objects support. For example:

HKCR\Word.Document.6\DocObject = 0
HKCR\CLSID\{<CLSID for Word Document>} = Microsoft Word 7.0 Document
HKCR\CLSID\{<CLSID for Word Document>}\DocObject = 0

(HKCR is short for HKEY_CLASSES_ROOT.)

The DocObject subkey should appear under both the server's ProgID and its CLSID. The value of the "DocObject" key indicates whether the DocObject can create multiple views and whether it can accept complex rectangles. See *IOleDocument::GetDocMiscStatus* for more information.

The DocObject must also use the "DefaultExtension" key to register the default extension used by its files along with a descriptive string that can be used in a File Open or File Save As dialog. For example:

\<CLSID for Word Document>\DefaultExtension=.doc, Word Documents (*.doc)

Finally, if the object supports the *IPrint* interface, it must register the "Printable" key. For example:

\<CLSID...>\Printable

Limiting Embedding Support

All DocObjects will be embeddable due to the fact that they implement all the relevant interfaces for OLE Documents (*IOleObject, IDataObject, IPersistFile*, and *IPersistStorage*). However, they can choose to limit the embedding functionality they support. This can be done as follows:

- Do not register the "Insertable" key to prevent compound document containers from listing the document object class in the Insert Object dialog.[4]
- Do not offer "Embed Source" or "Embedded Object" formats in data exchange operations. This prevents the object from being pasted into compound document containers.
- Set the OLEMISC_CANTLINKINSIDE bit in your MiscStatus key of the registry to prevent linking to embedded DocObjects.
- Set the OLEMISC_ICONICONLY bit to force the document to appear as an icon in any container that might

[4] **Also do not register a "\protocol\StdFileEditing\server" to prevent inclusion in an OLE 1 container's Insert Object dialog.**

still receive the object through the Insert From File dialog (an option in Insert Object) or when the file is dropped on a container from the system shell. Because it is only displayed as an icon, there is no need to worry about generating metafiles nor in handling *IOleObject::SetExtent* calls, etc.

Becoming a DocObject Container

This section discusses issues related to container or host side implementation of the Document Objects architecture. It goes without saying that a container supports the necessary interfaces as described in the architecture. However, there are a number of other considerations:

1. Storage provisions and user interface
2. Creation and initialization of a DocObject
3. Activation of a DocObject
4. DocObject saving and shutdown
5. Support for other OLE features, completeness of interface implementations

The following sections describe each of these topics in more detail. These are the core pieces of a DocObject container that require more comment than is found elsewhere in this specification, and the following discussion is not intended to touch on every container-side detail. As such, specific items like Help menu merging and command targets are not described here and are left for sample code to demonstrate.

Storage Provisions and User Interface

A DocObject container is generally a container that manages multiple "documents" (from the user perspective) in a single data store of some kind. Now, that data store could be something as complex as an entire file system, or it could be something simple like an individual compound file. In general, the container's methods for dealing with the ultimate storage of documents edited as DocObjects will in many ways determine the type of user interface that the container supports.

The Office Binder, for example, uses a single "Binder" file, an OLE Compound File, as its own data store. Within that single Binder file, the Binder can embed any number of other documents as "sections" in the binder. Technically speaking, while the Binder file itself is a single root instance of *IStorage*, each section is then given the *IStorage* of a sub-storage within the root. Each embedded DocObject is handed this sub-storage pointer through *IPersistStorage::InitNew* or *IPersistStorage::Load* (either at creation or reloading time, respectively) and stores all of its data directly in that storage.

What the Office Binder does for a user interface, then, is provide a left-hand pane that displays the "documents" or sections in the binder, activating them one at a time in the right-hand pane as if they were being opened in their respective applications. However, one never leaves the binder paradigm as one changes from section to section. Each so-called document is just a sub-storage in the entire binder.

Now a container that browses a file system, on the other hand, will see the whole file system, or the World Wide Web for that matter, as a single "file" or "binder" in which are found many individual documents. This kind of container would have the browsing UI in the left hand pane and would individually activate DocObjects within a viewing pane of that browser. In this case the *IStorage* handed to each DocObject is the root *IStorage* for the entire document on the file system itself.

One must not confuse the use of an *IStorage* in the DocObjects architecture with the use of streams to save and re-load view states through *IOleDocumentView::SaveViewState* and *IOleDocumentView::ApplyViewState* as described in more detail below.

Creation and Initialization

However a container wishes to create an embedded DocObject is up to that container. This will generally involve one of the OLE API functions *OleCreate, OleCreateFromData, OleCreateFromFile*, and *OleLoad. OleCreate*, of course, is used to create a new, uninitialized DocObject—when that object is activated the user starts with a clean slate. *OleCreateFromData* and *OleCreateFromFile*, on the other hand, create new instances of objects with a state initialized from either the contents of a data object (clipboard, drag and drop, etc.) or from the contents of a file, respectively. Once a DocObject is saved to its *IStorage* via *OleSave*, it can then be reloaded with *OleLoad*, of course.

The full initialization sequence for a DocObject will depend on the exact nature of the container. As a minimum, however, it will involve these steps after creation or loading:

1. *IPersistStorage::InitNew* (create) or *IPersistStorage::Load* (reload)
2. *IOleObject::SetClientSite*
3. *IOleObject::Advise*

These three calls will initialize the DocObject and set up communication between it and the container's *IOleClientSite* and *IAdviseSink* interfaces. Nothing more is essential, although containers that display something like an iconic rendering of the object may also include calls to *IViewObject2* members such as *GetExtent* and *SetAdvise*.

Note that as described in the server section above, the container's call to *IOleObject::SetClientSite* will generate a *QueryInterface* call to the container for *IOleDocumentSite*. The object then uses this interface during activation, which is the next topic.

Activation

Activating a DocObject is largely just a matter of calling *IOleObject::DoVerb(OLEIVERB_SHOW, ...)* then responding to *IOleDocumentSite::ActivateMe*. OLEIVERB_SHOW is generally the most appropriate activation verb here, but OLEIVERB_PRIMARY and OLEIVERB_UIACTIVATE are also allowable. OLEIVERB_OPEN isn't recommended as highly because it implies separate-window activation instead of an in-place activation.

Activation of a DocObject is almost entirely self-contained within *IOleDocumenSite::ActivateMe* whose implementation generally appears as follows:

```
STDMETHODIMP CImpIOleDocumentSite::ActivateMe(IOleDocumentView *pView)
    {
    RECT                rc;

    /*
     * If we're passed a NULL view pointer, then try to get one from
     * the document object.
     */
    if (NULL==pView)
        {
        IOleDocument *pDoc;

        if (FAILED(m_pSite->m_pObj->QueryInterface(IID_IOleDocument
            , (void **)&pDoc)))
            return E_FAIL;

        if (FAILED(pDoc->CreateView(m_pSite->m_pImpIOleIPSite, NULL
            , 0, &pView)))
            return E_OUTOFMEMORY;
        }
    else
        {
        //Make sure that the view has our client site
        pView->SetInPlaceSite(m_pSite->m_pImpIOleIPSite);

        //We're holding onto the pointer, so AddRef it.
        pView->AddRef();
        }

    //Remember the type of object we have and the view pointer
    m_pSite->m_fDocObj==TRUE;
    m_pSite->m_pIOleDocView=pView;

    //This sets up toolbars and menus first
    pView->UIActivate(TRUE);

    //Set the window size sensitive to new toolbars
    GetClientRect(m_pSite->m_hWnd, &rc);
    pView->SetRect(&rc);
    //Makes it all visible
    pView->Show(TRUE);

    return NOERROR;
    }
```

This code is taken from a working DocObject container and demonstrates the proper sequence of operations for DocObject activation:

1. If *ActivateMe* is passed an *IOleDocumentView* pointer, then call *IOleDocumentView::SetInPlaceSite* followed by *AddRef* if you're holding onto the pointer (which is generally the case). Otherwise query the document object itself for *IOleDocument* and call *IOleDocument::CreateView* passing in the container's *IOleInPlaceSite* pointer. In both cases you'll end up with an *IOleDocumentView* pointer for the DocObject's view that should be released when the container no longer needs it.
2. Activate the DocObject view by calling *IOleDocumentView::UIActivate(TRUE)* which will cause it to perform menu merging, toolbar negotiation, and re-parent its display window to the window returned through *IOleInPlaceSite::GetWindow*. Part of the toolbar negotiation sequence should be for the container to remember exactly how much border space is taken up, resizing any client-area windows in the container to account for this space.
3. Call *IOleDocumentView::SetRect* (or *SetRectComplex* depending on the container) to tell the view exactly how much space to occupy in its parent. If the container manages a client-area window as the code sample above is doing, then this rectangle is simply the client area of that window. Note that this step is important to do *after* calling *UIActivate* because the container would otherwise send the view the wrong dimensions that wouldn't account for toolbar space.
4. Call *IOleDocumentView::Show(TRUE)* to make the DocObject visible. This is the last step because the DocObject view knows exactly what space it occupies and all its other tools are there.

While the DocObject remains active, it is also imperative that the container fulfill a few other requirements, some of which come from standard in-place activation rules:

1. Call *IOleInPlaceActiveObject::ResizeBorder* when the container frame is resized so the object can resize its toolbars appropriately.
2. Call *IOleDocumentView::SetRect* whenever the window used for the DocObject parent is resized. This might be the frame window, a client-area window, or a document window (in an MDI container). *SetRect* tells the DocObject to resize its view to fully occupy the parent window's client area.
3. Implement *IOleInPlaceFrame::SetStatusText* if the container has a toolbar.
4. Call *IOleInPlaceActiveObject::TranslateAccelerator* from the container's message loop.
5. Detect the F1 key to enter context-sensitive help mode as well as ESC to leave it, calling *IOleInPlaceActiveObject::ContextSensitiveHelp* with TRUE and FALSE, respectively.
6. Handle WM_SETFOCUS to the frame window by setting focus to the window returned from *IOleInPlaceActiveObject::GetWindow*.

All of these bits other than step 2 are standard in-place activation requirements.

Saving and Shutdown

When a DocObject is closed, the container should ensure that its data is saved as it would with any other embedded object. That is, the container must handle *IOleClientSite::SaveObject* in which it generates a call to the object's *IPersistStorage::Save*, usually through *OleSave*, followed by *IPersistStorage::SaveCompleted* and an *IStorage::Commit* if transactioned storage is being employed.

When the container wishes to close the object entirely, that is, unload it completely (with or without saving), then the container should first call *IOleInPlaceObject::InPlaceDeactivate*, *IOleObject::Close*, followed by *Release* calls on all interface pointers that the container is holding. When the last reference count is released, the DocObject will delete itself and its server will shut down as appropriate.

Again, all of this is standard for standard embedding scenarios in OLE Documents.

Support for Other OLE Features and Completeness of Interface Implementations

As described in the previous section, DocObject servers will never generate calls to various members of the *IOleClientSite* and *IOleInPlaceSite* interfaces, such as:

- *IOleClientSite::GetMoniker*
- *IOleClientSite::GetContainer*
- *IOleClientSite::RequestNewObjectLayout*
- *IOleClientSite::OnShowWindow*
- *IOleClientSite::ShowObject*
- *IOleInPlaceSite::OnPosRectChange*
- *IOleInPlaceSite::Scroll*
- *IOleInPlaceSite::ContextSensitiveHelp* (if container has no support for this)

Therefore strictly DocObject containers can simply return E_NOTIMPL from these members. In addition, most of the *IAdviseSink* members need no implementation, with *IAdviseSink::OnClose* being the only one of probable interest; in some cases a container may not need *IAdviseSink* at all, and thus would never need to call *IOleObject::Advise* (or *IViewObject::SetAdvise* when the container doesn't display anything for the object visually).

All other members of *IOleClientSite* and *IOleInPlaceSite*, as well as those in *IOleInPlaceFrame* require some implementation which is sometimes considerable and at other times nothing more than a return of NOERROR (such as *IOleInPlaceSite::CanInPlaceActivate*).

Of course, the container may support more than just DocObjects—it might also support normal OLE compound document linking and embedding in which case it will completely implement these interfaces as specified for OLE Documents. The container may also support OLE Controls in which case it would have *IDispatch*, event handlers, *IOleControlSite*, and so on. It should be noted, however, that DocObjects do not interfere with support for these other types of objects, provided that the container maintains a variable (like *m_fDocObj* as described in the activation section above) that tells the rest of its code that certain operations won't be needed when a DocObject is in use.

Finally, there is also the separate-window activation model available through *IOleDocumentView* which can be employed as a container sees fit. Support for that model is not a requirement for all DocObject containers, however.

Document Objects Interface Reference

This section lists all interfaces, related structures, and related enumerations that are defined by this architecture. The section has a detailed description of interface member functions and their arguments.

The IOleDocument *Interface*

By implementing this interface alongside other interfaces relating to OLE Documents, an object indicates its ability to act as a "document object." Through this interface, a container for DocObjects can ask the object to create views of itself as well as to enumerate those views and to retrieve MiscStatus bits related to the document object.

IDL:

```
[
uuid(B722BCC5-4E68-101B-A2BC-00AA00404770)
    , object, pointer_default(unique)
]
interface IOleDocument : IUnknown
    {
    HRESULT CreateView([in] IOleInPlaceSite *pIPSite, [in] IStream *pstm
        , [in] DWORD dwReserved, [out] IOleDocumentView **ppView);
    HRESULT GetDocMiscStatus([out] DWORD *pdwStatus);
    HRESULT EnumViews([out] IEnumOleDocumentViews **ppEnum
        , [out] IOleDocumentView **ppView);
    }
```

IOleDocument::CreateView

HRESULT IOleDocument::CreateView([in] IOleInPlaceSite *pIPSite, [in] IStream *pstm
, [in] DWORD dwReserved, [out] IOleDocumentView **ppView)

Ask the document object to create a new view sub-object, returning that view object's *IOleDocumentView* interface pointer. Optionally this call can also initialize the view from the contents a given stream. A container calls this function to both create new views as well as to reload previously saved views. The view must wait for calls to either *IOleDocumentView::Show* or *IOleDocumentView::UIActivate* before showing itself.

Argument	Typ	Description
pIPSite	*IOleInPlaceSite* *	A pointer to the container's "view site" object associated with the new view. May be NULL in which case the caller must initialize the view with a call to *IOleDocumentView::SetInPlaceSite*.
psim	*IStream* *	A pointer to the stream from which the view should initialize itself. If NULL, then this function creates a new view with a default state.
dwReserved	*DWORD*	Reserved for future use. Must be zero.[5]
ppView	*IOleDocumentView* *	Address of the variable to receive the interface pointer to the new view. If *CreateView* succeeds, the caller is responsible for calling *Release* through this pointer when the view object is no longer needed.

[5] **In the future this parameter could be used to specify the type of view that needs to be created. Currently there are no defined values for this argument.**

Return Value	Meaning
S_OK	The view was created successfully.
E_POINTER	The address in *ppView* is NULL.
E_OUTOFMEMORY	There is not enough memory to create the new view.
E_UNEXPECTED	An unknown error occurred.
E_FAIL	This document object only supports a single view which has already been created.

Comments:
This function must be completely implemented in any document object; therefore E_NOTIMPL is not an acceptable return code.

As with all new interface pointers, *CreateView* calls *AddRef* on the pointer in **ppView* before returning. The caller is responsible for calling *Release* through this pointer when it is no longer needed.

If *pIPSite* is non-NULL, then the document object should pass the pointer to the new view through *IOleDocumentView::SetInPlaceSite*. If NULL, the caller is responsible for this same call. In addition, if *pstm* is non-NULL, then the object should initialize the view object by passing *pstm* to *IOleDocumentView::ApplyViewState*.

IOleDocument::GetDocMiscStatus

HRESULT IOleDocument::GetDocMiscStatus([out] DWORD *pdwStatus)

Returns miscellaneous status bits describing the document object, such as whether the object can create multiple views and accept complex rectangles.[6] These values are also stored in the registry as the value of the "DocObject" key:

```
typedef enum
    {
    DOCMISC_CANCREATEMULTIPLEVIEWS    = 1, //Object supports multiple views
    DOCMISC_SUPPORTCOMPLEXRECTANGLES = 2, //IOleDocumentView::SetRectComplex
is supported
    DOCMISC_CANTOPENEDIT             = 4, //IOleDocumentView::Open is
unsupported
    DOCMISC_NOFILESUPPORT            = 8  //Object does not support file
read/write
    } DOCMISC;
```

The bits DOCMISC_CANTOPENEDIT, DOCMISC_NOFILESUPPORT need further explanation. There can be objects which can only be embedded, can only be in-place activated, and which do not have files of their own, regardless of whether they are implemented as in-process or local servers. Objects which have limited UI for activation purposes should set DOCMISC_CANTOPENEDIT. Those that only support *IPersistStorage* as a persistence mechanism should specify DOCMISC_NOFILESUPPORT. Otherwise and object must also implement *IPersistFile* implementation.

If an object desires none of these status bits it must return a zero in **pdwStatus*.

Argument	Type	Description
pdwStatus	*DWORD **	The address of the variable to receive the status bits about this document object.

Return Value	Meaning
S_OK	The status bits were returned successfully.
E_POINTER	The address in *pdwStatus* is NULL.

[6] **One rectangle each for view, horizontal scroll bar, vertical scroll bar and size box. See *IOleDocumentView::SetRectComplex*.**

Comments:
This function must be completely implemented in any document object even if a zero is returned; therefore E_NOTIMPL is not an acceptable return code.

IOleDocument::EnumViews

**HRESULT IOleDocument::EnumViews([out] IEnumOleDocumentViews **ppEnum
, [out] IOleDocumentView **ppView)**

Creates an enumerator object that enumerates the IOleDocumentView interface pointers of the views of the document object. The enumerator supports the interface *IEnumOleDocumentViews*, a pointer to which is returned in **ppEnum*. An object that supports only a single view (that is, DOCMISC_CANCREATEMULTIPLEVIEWS is not specified through *IOleDocument::GetMiscStatus*) does not create an enumerator but instead returns the single view pointer through **ppView*.

Argument	Type	Description
ppEnum	*IEnumOleDocumentViews* **	The address of the variable to receive the interface pointer of the enumerator.
ppView	*IOleDocumentView* **	The address of the variable to receive the interface pointer of a single view.

Return Value	Meaning
S_OK	If the object supports multiple views, then **ppEnum* contains the enumerator pointer. Otherwise **ppEnum* is NULL and **ppView* contains the interface pointer to the single view..
E_POINTER	The address in *ppEnum* or *ppView* is invalid. The caller must pass pointers for both arguments.
E_OUTOFMEMORY	The enumerator could not be created because there is insufficient memory.

Comments:
This function must be completely implemented in any document object; therefore E_NOTIMPL is not an acceptable return code.

The IEnumOleDocumentViews *Interface*

A document object can be asked to enumerate its views through *IOleDocument::EnumViews*. The resulting enumerator returned from this member implements the interface *IEnumOleDocumentViews* through which a client can access all the individual view sub-objects supported within the document object itself, where each view implements *IOleDocumentView*.

Therefore *IEnumOleDocumentViews* is a standard enumerator interface typed for *IOleDocumentView* *.

IDL:

```
[
uuid(B722BCC8-4E68-101B-A2BC-00AA00404770)
    , object, pointer_default(unique)
]
interface IEnumOleDocumentViews : IUnknown
    {
    HRESULT Next([in] ULONG cViews
        , [out, max_is(cViews)] IOleDocumentView **rgpView
        , [out] ULONG *pcFetched);

    HRESULT Skip([in] ULONG cViews);
    HRESULT Reset(void);
    HRESULT Clone([out] IEnumOleDocumentViews **ppEnum);
    }
```

IEnumOleDocumentViews::Next

HRESULT IEnumOleDocumentViews::Next([in] ULONG cViews , [out, max_is(cViews)] IOleDocumentView **rgpView, [out] ULONG *pcFetched);

Enumerates the next *cViews* elements in the enumerator's list, returning them in *rgpView* along with the actual number of enumerated elements in *pcFetched*. The caller is responsible for calling *IOleDocumentView::Release* through each pointer returned in *rgpView*.

Argument	Typ	Description
cViews	*ULONG*	Specifies the number of *IOleDocumentView** values to return in the array pointed to by *rgpView*. This argument must be 1 if *pcFetched* is NULL.
rgpView	*IOleDocumentView* *	A pointer to a caller-allocated *IOleDocumentView* * array of size *cViews* in which to return the enumerated document views. The caller is responsible for calling *IOleDocumentView::Release* through each pointer enumerated into the array once this method returns successfully. If *cViews* is greater than 1 the caller must also pass a non-NULL pointer passed to *pcFetched* to know how many pointers to release.
pcFetched	*ULONG*	A pointer to the variable to receive the actual number of document views enumerated in *rgpView*. This argument can be NULL in which case the *cViews* argument must be 1.

Return Value	Meaning
S_OK	The requested number of elements has been returned and **pcFetched* (if non-NULL) is set to *cViews* if
S_FALSE	The enumerator returned fewer elements than *cViews* because there were not that many elements left in the list.. In this case, unused elements in *rgpView* in the enumeration are not set to NULL and **pcFetched* holds the number of valid entries, even if zero is returned.
E_POINTER	The address in *rgpView* is not valid (such as NULL)

E_INVALIDARG	The value of *cViews* is not 1 when *pcFetched* is NULL; or the value of *cViews* is zero.
E_UNEXPECTED	An unknown error occurred.
E_OUTOFMEMORY	There is not enough memory to enumerate the elements.

Comments:
E_NOTIMPL is not allowed as a return value. If an error value is returned, no entries in the *rgpView* array are valid on exit and require no release.

IEnumOleDocumentViews::Skip

HRESULT IEnumOleDocumentViews::Skip([in] ULONG cConnections);

Instructs the enumerator to skip the next *cViews* elements in the enumeration such that the next call to *IEnumOleDocumentViews::Next* will not return those elements.

Argument	Type	Description
cViews	*ULONG*	Specifies the number of elements to skip in the enumeration.

Return Value	Meaning
S_OK	The number of elements skipped is *cViews*.
S_FALSE	The enumerator skipped fewer than *cViews* because there were not that many left in the list. The enumerator will, at this point, be positioned at the end of the list such that subsequent calls to *Next* (without an intervening *Reset*) will return zero elements.
E_INVALIDARG	The value of *cViews* is zero, which is not valid.
E_UNEXPECTED	An unknown error occurred.

IEnumOleDocumentViews::Reset

HRESULT IEnumOleDocumentViews::Reset(void);

Instructs the enumerator to position itself back to the beginning of the list of elements.

Argument	Type	Description
NA	NA	NA

Return Value	Meaning
S_OK	The enumerator was successfully reset to the beginning of the list.
S_FALSE	The enumerator was not reset to the beginning of the list.
E_UNEXPECTED	An unknown error occurred.

Comments:
There is no guarantee that the same set of elements will be enumerated on each pass through the list: it depends on the collection being enumerated. It is too expensive for some collections, such as files in a directory, to maintain this condition.

IEnumOleDocumentViews::Clone

HRESULT IEnumOleDocumentViews::Clone([out] IEnumOleDocumentViews **ppEnum);

Creates another view enumerator with the same state as the current enumerator, which iterates over the same list. This makes it possible to record a point in the enumeration sequence in order to return to that point at a later time.

Argument	Type	Description
ppEnum	*IEnumOleDocumentViews***	The address of the variable to receive the *IEnumOleDocumentViews* interface pointer to the newly created enumerator. The caller must release this new enumerator separately from the first enumerator.

Return Value	Meaning
S_OK	Clone creation succeeded.
E_NOTIMPL	Cloning is not supported for this enumerator.
E_POINTER	The address in *ppEnum* is not valid (such as NULL)
E_UNEXPECTED	An unknown error occurred.
E_OUTOFMEMORY	There is not enough memory to create the clone enumerator.

The IOleDocumentSite *Interface*

By implementing this interface on an client site alongside other client site interfaces required by OLE Documents, a container indicates its support for document object activation to any such objects associated with this site. The interface allows a document object to ask the container to activate it as a document instead of as an in-place embedded object. The document object can alternately specify which view to activate.

The view site encapsulates the view port (the HWND and a rectangle in that HWND) and the frame context of the view port. There can be multiple view ports in a single window. A view site is attached to a view through the *pIPSite* argument of *IOleDocument::CreateView* or through *IOleDocumentView::SetInPlaceSite.*

IDL:

```
[
uuid(B722BCC7-4E68-101B-A2BC-00AA00404770)
    , object, pointer_default(unique)
]
interface IOleDocumentSite : IUnknown
    {
    HRESULT ActivateMe([in] IOleDocumentView *pViewToActivate);
    };
```

IOleDocumentSite::ActivateMe

HRESULT IOleDocumentSiteActivateMe([in] IOleDocumentView *pViewToActivate)

When a document object is asked to in-place activate through *IOleObject::DoVerb*, a document object bypasses the normal in-place activation sequence of OLE Documents and instead calls *IOleDocumentSite::ActivateMe* to become active as a document. This should be done in the OLEIVERB_OPEN, OLEIVERB_SHOW, OLEIVERB_INPLACEACTIVATE, and OLEIVERB_UIACTIVATE cases.

The document object can specify which view to activate by passing that view's *IOleDocumentView* pointer in *pViewToActivate*. The container in this case will proceed and activate that view through that pointer. Otherwise, the container calls the object's *IOleDocument::CreateView* to obtain the view it wishes to activate.

Argument	**Type**	**Description**
pViewToActivate	*IOleDocumentView* **	If non-NULL, specifies the view to bring forward. The caller does not call *AddRef* on this pointer before passing it in—if the receiver wishes to hold the pointer outside of this member function it must call *pViewToActivate->AddRef();*

Return Value	**Meaning**
S_OK	The container activated the view successfully.
E_OUTOFMEMORY	*pViewToActivate* is NULL and the container's call to *IOleDocument::CreateView* failed with E_OUTOFMEMORY.
E_FAIL	Another error occurred in either view creation or activation.

Comments:
This function must be completely implemented in a container; therefore E_NOTIMPL is not an acceptable return code.

The IOleDocumentView *Interface*

Each view of a document object is a sub-object that implements *IOleDocumentView* alongside *IOleInPlaceObject*, *IOleInPlaceActiveObject*, and other optional interfaces like *IPrint* and *IOleCommandTarget*. This interface provides all the necessary operations for a container to manipulate, manage, and activate a view.

IDL:

```
[
uuid(B722BCC6-4E68-101B-A2BC-00AA00404770)
    , object, pointer_default(unique)
]
interface IOleDocumentView : IUnknown
    {
    //import "unknwn.idl";

    HRESULT SetInPlaceSite([in] IOleInPlaceSite *pIPSite);
    HRESULT GetInPlaceSite([out] IOleInPlaceSite **ppIPSite);
    HRESULT GetDocument([out] IUnknown **ppunk);
    [input_sync] HRESULT SetRect([in] LPRECT prcView);
    HRESULT GetRect([out] LPRECT prcView);
    [input_sync] HRESULT SetRectComplex([in] LPRECT prcView
        , [in] LPRECT prcHScroll, [in] LPRECT prcVScroll
        , [in] LPRECT prcSizeBox);
    HRESULT Show ([in] BOOL fShow);
    HRESULT UIActivate([in] BOOL fUIActivate);
    HRESULT Open(void);
    HRESULT CloseView([in] DWORD dwReserved);
    HRESULT SaveViewState([in] IStream *pstm);
    HRESULT ApplyViewState([in] IStream *pstm);
    HRESULT Clone([in] IOleInPlaceSite *pIPSiteNew
        , [out] IOleDocumentView **ppViewNew);
    }
```

The members *SetInPlaceSite* and *GetInPlaceSite* manage the *IOleInPlaceSite* interface pointer for the container's view site associated with this view. The semantics of *SetInPlaceSite* are encompassed in the *pIPSite* argument of *IOleDocument::CreateView.*

GetDocument provides access to the *IUnknown* pointer of the document object that owns this view.

The *SetRect* and *GetRect* members manage the simple rectangle that the view will occupy in the container. *SetRectComplex* allows the container to specify not only the simple rectangle but also the spaces that should be occupied by the view's scrollbars and size box. An view specifies whether it understands *SetRectComplex* through the DOCMISC_SUPPORTCOMPLEXRECTANGLES status bit (see *IOleDocument::GetMiscStatus*).

The view's visual state is managed through the pair *Show* and *UIActivate* as well as *Open*. *Show* instructs the view to activate or deactivate itself in-place; when the view is active, *UIActivate* instructs the view to activate or deactivate its user interface elements such as menus, toolbars, and accelerators. *Show* and *UIActivate* in this interface are thus equivalents of the *IOleInPlaceObject* members of *InPlaceActivate, InPlaceDeactivate, UIActivate, and UIDeactivate* that are used for control of an in-place embedding.

The *Open* member, on the other hand, works with activation in a separate window (as happens with embeddings in OLE Documents when in-place is not supported). DocObjects marked with DOCMISC_CANTOPENEDIT (see *IOleDocument::GetMiscStatus*) do not support this form of activation. If support is present, however, *Open* instructs the view to activate in a separate window similar to *IOleObject::DoVerb(OLEIVERB_OPEN).* At this point *Show* instructs the view to show and hide this window.

In all cases, *CloseView* instructs the view to deactivate the view, destroying any separate window and releasing the view site pointer passed previously to *IOleDocumentView::SetInPlaceSite*. This functionality is similar to that described for *IOleObject::Close.*

A view's internal state can be saved to a stream through *SaveViewState* and later reloaded from a stream with the same contents through *ApplyViewState*. The semantics of *ApplyViewState* are encompassed in the *pstm* argument of *IOleDocument::CreateView*.

Finally, a container can create a duplicate view object to the current one with *Clone*.

IOleDocumentView::SetInPlaceSite

HRESULT IOleDocumentView::SetInPlaceSite([in] IOleInPlaceSite *pIPSite)

Associates a view site object with this view. If this member is called and the view already has an associated view site, the view must first deactivate itself in that site, release that site, then remember the new pointer if that pointer is non-NULL (save the value and call *AddRef* on the pointer). The container will tell the view when to activate itself in the new site.

Argument	Type	Description
pIPSite	*IOleInPlaceSite* *	The interface pointer of the site to associate with this view. Can be NULL in which case the view loses all association with the container.

Return Value	Meaning
S_OK	The site was successfully associated (or disassociated if *pIPSite* is NULL)
E_FAIL	Another error occurred.

Comments:
This function must be completely implemented in a view; therefore E_NOTIMPL is not an acceptable return code.

IOleDocumentView::GetInPlaceSite

HRESULT IOleDocumentView::GetInPlaceSite([out] IOleInPlaceSite **ppIPSite)

Returns the most recent *IOleInPlaceSite* pointer passed to *SetInPlaceSite*, or NULL if *SetInPlaceSite* has not yet been called. The view will call *AddRef* on this pointer before returning it, thus the caller must later call *Release*.

Argument	Type	Description
ppIPSite	*IOleInPlaceSite* **	The address in which to return the current view site interface pointer associated with this view object. The caller becomes responsible for this pointer.

Return Value	Meaning
S_OK	The site was successfully returned. The caller must call *Release* through this pointer when it is no longer needed.
E_FAIL	Another error occurred.

Comments:
This function must be completely implemented in a view; therefore E_NOTIMPL is not an acceptable return code.

IOleDocumentView::GetDocument

HRESULT IOleDocumentView::GetDocument([out] IUnknown **ppunk)

Returns the *IUnknown* interface pointer of the document object that owns this view. As a document owning the view must always exist, this function will always succeed, calling *AddRef* on the pointer stored in **ppunk* before returning.

Argument	Type	Description
ppunk	*IUnknown* **	The address in which to return the *IUnknown* pointer of the document object that owns this view. The caller becomes responsible for this pointer.

Return Value	Meaning
S_OK	The document object's interface pointer was successfully returned. This is the only valid return code for this function.

IOleDocumentView::SetRect

[input_sync] HRESULT IOleDocumentView::SetRect([in] LPRECT prcView)

Sets the rectangular coordinates of the view port in the client coordinates of the view window (the window is obtained through *IOleInPlaceSite::GetWindow*). The view must resize itself to view the new coordinates.

This member function is defined with the [input_sync] attribute, hence the implementing object cannot yield or make another non input_sync RPC call while executing this method.

Argument	Type	Description
prcView	*LPRECT*	Points to a RECT structure containing the coordinates of the view port in the client coordinates of the view window.

Return Value	Meaning
S_OK	The view was successfully resized to the rectangle.
E_FAIL	Some other critical error occurred that prevented resizing to occur.

Comments:
This function must be completely implemented in a view; therefore E_NOTIMPL is not an acceptable return code.

IOleDocumentView::GetRect

HRESULT IOleDocumentView::GetRect([out] LPRECT prcView)

Returns the rectangular coordinates of the view port in the client coordinates of the view window, as was last specified through *IOleDocumentView::SetRect* or *IOleDocumentView::SetRectComplex* .

Argument	Type	Description
prcView	*LPRECT*	Points to a RECT structure to receive the current view coordinates.

Return Value	Meaning
S_OK	The view was successfully resized to the rectangle.
E_UNEXPECTED	This view has not yet seen a call to *IOleDocumentView::SetRect* or *IOleDocumentView::SetRectComplex*, thus it has no rectangle to return.

Comments:
This function must be completely implemented in a view; therefore E_NOTIMPL is not an acceptable return code.

IOleDocumentView::SetRectComplex

[input_sync] HRESULT IOleDocumentView::SetRectComplex([in] LPRECT prcView, [in] LPRECT prcHScroll, [in] LPRECT prcVScroll, [in] LPRECT prcSizeBox)

Sets the rectangular coordinates of the view port, horizontal and vertical scroll bars, and the size box. This method typically gets used by the view frames which have a workbook metaphor. However, not all DocObjects support these detailed specifications; those that do mark themselves with DOCMISC_SUPPORTCOMPLEXRECTANGLES as described in *IOleDocument::GetMiscStatus*. DocObjects that do not support this member can return E_NOTIMPL.

Within this member, the view should resize itself according to *prcView* and fit its scrollbars and size box to the areas described in *prcHScroll*, *prcVScroll*, and *prcSizeBox*, respectively.

This member function is defined with the [input_sync] attribute, hence the implementing object cannot yield or make another non input_sync RPC call while executing this method.

Argument	Type	Description
prcView	[in] LPRECT	Points to a RECT structure containing the coordinates of the view port in client coordinates of the view window.
prcHScroll	[in] LPRECT	Points to a RECT structure containing the coordinates of the horizontal scroll bar in client coordinates of the view window.
prcVScroll	[in] LPRECT	Points to a RECT structure containing the coordinates of the vertical scroll bar in client coordinates of the view window.
prcSizeBox	[in] LPRECT	Points to a RECT structure containing the coordinates of the size box in client coordinates of the view window.

Return Value	Meaning
S_OK	The view was successfully resized to the rectangle.
E_NOTIMPL	The document object that owns this view does not support complex rectangle specifications.
E_FAIL	Some other critical error occurred that prevented resizing of the view or placement of the scrollbars and size box.

IOleDocumentView::Show

HRESULT Show ([in] BOOL fShow)

Instructs a view to in-place activate or in-place deactivate itself as described in the following pseudo-code:

```
if (fShow)
    {
    in-place activate the view but do not UI activate it.
    Show the view window.
    {
else
    {
    call IOleDocumentView::UIActivate(FALSE) on this view
    Hide the view window
    }
```

Argument	Type	Description
fShow	BOOL	TRUE instructs the view to show itself, FALSE instructs the view to hide itself.

Return Value	Meaning
S_OK	The view was successfully shown or hidden.
E_OUTOFMEMORY	There was not enough memory to activate or hide the view.
E_FAIL	Some other critical error occurred that prevented activation or hiding.
E_UNEXPECTED	This member was called before a call to *IOleDocumentView::SetInPlaceSite.*

Comments:
All views of a document object must at least support the in-place activation mode, therefore E_NOTIMPL is not allowed as a return value.

IOleDocumentView::UIActivate

HRESULT IOleDocumentView::UIActivate([in] BOOL fUIActivate)

Instructs the view to activate or deactivate its user interface elements (menus, toolbars, accelerators) as described in the following pseudo-code:

```
if (fActivate)
    {
    UI activate the view (do menu merging, show frame level tools, process
accelerators)
    Take focus, and bring the view window forward.
    }
else
    call IOleInPlaceObject::UIDeactivate() on this view
```

The view may, and should, participate in extended Help menu merging if it desires.

Argument	Type	Description
fActivate	BOOL	TRUE instructs the view to activate its UI, FALSE instructs the view to deactivate its UI.

Return Value	Meaning
S_OK	The view's UI was successfully activated or deactivated.
E_OUTOFMEMORY	There was not enough memory to activate the UI elements.
E_FAIL	Some other error occurred that prevented success.
E_UNEXPECTED	This member was called before a call to *IOleDocumentView::SetInPlaceSite.*

Comments:
All views of a document object must at least support the in-place activation mode, therefore E_NOTIMPL is not allowed as a return value.

IOleDocumentView::Open

HRESULT IOleDocumentView::Open(void)

Asks the view to display itself in a separate popup window with semantics equivalent to *IOleObject:;DoVerb(OLEIVERB_OPEN).* If the document object specified DOCMISC_CANTOPENEDIT through *IOleDocument::GetMiscStatus,* this call can return E_NOTIMPL. Otherwise implementation generally calls the view's own *IOleInPlaceObject::InPlaceDeactivate* after which the view shows its separate popup window and brings that window to the foreground.

Contrary to the normal in-place deactivation sequence for OLE Documents, a view *continues to hold* the *IOleInPlaceSite* pointer that it obtained in *IOleDocumentView::SetInPlaceSite* (likewise the view site continues to hold the view's interface pointers, obviously). This pointer is only released through *IOleDocumentView::SetInPlaceSite(NULL)* or in

IOleDocumentView::CloseView.

When the user closes the view's window (via File.Close), then the view should not shut itself down. Instead it should call *pIPSite->OnInPlaceActivate.* The view site then decides whether to UI activate the view at that time or at a later time.

When the container decides that the view window is no longer needed, it calls *IOleDocumentView::CloseView.* The view uses that call to determine when to release the site pointer and destroy the window.

If is legal for the container to call *IOleDocumentView::Show(FALSE)* when the view is in this Open mode. In this case the view hides its window. Similarly, *IOleDocumentView::Show(TRUE)* instructs the view to show the window again and bring it to the foreground.

Argument	Type	Description
NA	NA	NA

Return Value	Meaning
S_OK	The view successfully created its separate window.
E_OUTOFMEMORY	There was not enough memory to activate the view in a separate window.
E_FAIL	Some other error occurred that prevented success.
E_NOTIMPL	The document object that owns this view does not support separate window activation.
E_UNEXPECTED	This member was called before a call to *IOleDocumentView::SetInPlaceSite.*

IOleDocumentView::CloseView

HRESULT IOleDocumentView::CloseView([in] DWORD dwReserved)

Asks the view to close down and release its *IOleInPlaceSite* pointer obtained in *IOleDocumentView::SetInPlaceSite*. The container must call this method before it wants to delete the view (that is, release its last reference to the view). In general, implementation of this member will call *IOleDocumentView::Show (FALSE)* to hide the view if it's not already, then call *IOleDocumentView::SetInPlaceSite(NULL)* to deactivate itself and release the view site pointer.

Argument	Type	Description
dwReserved	*DWORD*	Reserved. Must be zero.

Return Value	Meaning
S_OK	The view successfully closed itself.

Comments:
Because *CloseView* is called when the container wishes to completely shut down the view, this member must be implemented and has no reason to fail.

IOleDocumentView::SaveViewState

HRESULT IOleDocumentView::SaveViewState([in] IStream *pstm)

Instructs the view to save its state into the given stream, where the state includes properties like the view type, zoom factor, insertion point, and so on. The container typically calls this function before deactivating the view. The stream can then later be used to reinitialize a view of the same document to this saved state through *IOleDocumentView::ApplyViewState.*

The view must write its CLSID as the first element in the stream according to the rules that apply to *IPersistStream*. Any cross-platform file format compatibility issues that apply to the document's storage representation also apply to this context.

Argument	Type	Description
pstm	[in] IStream *	ask the view to save the view state into this stream.

Argument	Type	Description
pstm	*IStream* *	The stream in which the view should save its state.

Return Value	Meaning
S_OK	The view successfully saved its state to the stream.
E_POINTER	The value in *pstm* is NULL.
E_NOTIMPL	This view has no meaningful state to save; this should be a rare case as most views will have at least some information.

IOleDocumentView::ApplyViewState

```
HRESULT IOleDocumentView::ApplyViewState([in] IStream *pstm)
```

Instructs a view to reinitialize itself according to the data in a stream that was previously written through *IOleDocumentView::SaveViewState*. Typically this function is called when the view is being displayed for first time after its instantiation. It is the responsibility of the view to validate the data in the view stream as the container does not attempt to interpret view state stream data in any way.

Argument	Type	Description
pstm	*IStream* *	The stream from which the view should load its state.

Return Value	Meaning
S_OK	The view successfully loaded its state from the stream.
E_POINTER	The value in *pstm* is NULL.
E_NOTIMPL	This view has no meaningful state that it would load; this should be a rare case as most views will have at least some information.

IOleDocumentView::Clone

```
HRESULT IOleDocumentView::Clone([in] IOleInPlaceSite *pIPSiteNew, [out] IOleDocumentView
    **ppViewNew)
```

Creates a duplicate view object with an identical internal state to the current view. This is useful for creating a new view with a different view port and view site but with the same view context as the view being cloned. Typically this will be used to implement the "Window-New window" functionality.

Argument	Type	Description
pipsiteClone	[in] IOleInPlaceSite *	pointer to the in-place site for the clone
ppviewClone	[out] IOleDocumentView **	the location where the pointer to the new view should be returned.

Argument	Type	Description
pIPSiteNew	*IOleInPlaceSite* *	The *IOleInPlaceSite* pointer of the view site to associate with the clone. The view being cloned should pass this to the new view's *IOleDocumentView::SetInPlaceSite* member. This can be NULL in which case the caller is responsible for calling *SetInPlaceSite* on this new view directly.
ppViewNew	*IOleDocumentView* *	The address of the variable to receive the pointer to the new view's *IOleDocumentView* interface. The caller is responsible for this pointer any must call *Release* through it when it is no longer needed.

Return Value	Meaning
S_OK	The view successfully cloned. The caller is responsible for the pointer in **ppViewNew.*
E_POINTER	The value in *ppViewNew* is NULL.
E_FAIL	The document object only supports one view. E_NOTIMPL can also be used.

The IPrint *Interface*

Any object that wishes to support programmatic printing can implement the *IPrint* interface. Through this interface a caller can tell the object to print, set the initial page number (for printing multiple documents together), and retrieve print-related information from the object:

IDL:

```
[
uuid(B722BCC9-4E68-101B-A2BC-00AA00404770)
    , object, pointer_default(unique)
]
interface IPrint : IUnknown
    {
    typedef [unique] IPrint *LPPRINT;

    typedef enum
        {
        PRINTFLAG_MAYBOTHERUSER        = 1,
        PRINTFLAG_PROMPTUSER           = 2,
        PRINTFLAG_USERMAYCHANGEPRINTER = 4,
        PRINTFLAG_RECOMPOSETODEVICE    = 8,
        PRINTFLAG_DONTACTUALLYPRINT    = 16,
        PRINTFLAG_FORCEPROPERTIES      = 32,
        PRINTFLAG_PRINTTOFILE          = 64
        } PRINTFLAG;

    typedef struct tagPAGERANGE
        {
        LONG nFromPage;
        LONG nToPage;
        } PAGERANGE;

    typedef struct tagPAGESET
        {
        ULONG cbStruct;
        BOOL  fOddPages;
        BOOL  fEvenPages;
        ULONG cPageRange;
        [size_is(cPageRange)] PAGERANGE rgPages[];
        } PAGESET;


    HRESULT SetInitialPageNum([in] LONG nFirstPage);
    HRESULT GetPageInfo([out] LONG *pnFirstPage, [out] LONG *pcPages);
    HRESULT Print([in] DWORD grfFlags, [in,out] DVTARGETDEVICE **pptd
        , [in,out] PAGESET **ppPageSet
        , [unique][in,out] STGMEDIUM *pstgmOptions
        , [in] IContinueCallback *pcallback, [in] LONG nFirstPage
        , [out] LONG *pcPagesPrinted, [out] LONG *pnLastPage);
    };

#define PAGESET_TOLASTPAGE ((WORD)(-1L))
```

The structures of this interface will be described first, followed by the member functions.

PAGERANGE Structure

Identifies a single range of pages. Note that is *nFromPage* is greater than *nToPage,* the pages are printed in the reverse order.

Member	Type	Description
nFromPage	*LONG*	The first page to print. The first page of a document is 1.
nToPage	*LONG*	The last page to print. A special value of PAGESET_TOLASTPAGE indicates that all the remaining pages should be printed.

PAGESET Structure

Identifies a series of page-ranges and optionally identifies only the even or odd pages as part of this PAGESET.

Member	Type	Description
cbStruct	*ULONG*	The number of bytes in this instance of the PAGESET structure. Must be a multiple of 4.
fOddPages	*BOOL*	If true, then only the odd-numbered pages in the page-set indicated by *rgPages* are to be printed.
fEvenPages	*BOOL*	If true, then only the even-numbered pages in the page-set indicated by *rgPages* are to be printed.
cPageRange	*ULONG*	The number of page-range pairs specified in *rgPages*.
rgPages	*PAGERANGE* *	Specifies the pages to be printed. The page ranges must be sorted in increasing order and non-overlapping. It is an error to attempt to print a page which does not exist.

PRINTFLAG Enumeration

A combination of values from PRINTFLAG is passed in as *grfFlags* to *IPrint::Print.*

Value	Description
PRINTFLAG_MAYBOTHERUSER	Specifies whether any interaction is permitted with the user at all. Unless this flag is set, no part of the printing process may interact with the user.
PRINTFLAG_PROMPTUSER	Only valid if PRINTFLAG_MAYBOTHERUSER is specified. Prompt the user for job-specific printing options using the normal print dialog for the object. Support for this option is required.
PRINTFLAG_USERMAYCHANGEPRINTER	Only valid if PRINTFLAG_PROMPTUSER is specified. Indicates that the user may change the printer to be printed to; in the absence of this flag, the user must print on the printer provided.
PRINTFLAG_RECOMPOSETODEVICE	Indicates that the object should attempt to recompose itself to the indicated target device. In the absence of this flag, the object should retain any existing compositional-device association that it may happen to presently have if at all possible.
PRINTFLAG_DONTACTUALLYPRINT	Carry out any indicated user-prompting and object-recomposing actions as indicated, but don't actually carry out the printing operation.
PRINTFLAG_PRINTTOFILE	The object should print to the file, name of which is passed through "portname" field of DVTARGETDEVICE.

IPrint::SetInitialPageNum

HRESULT IPrint::SetInitialPageNum([in] LONG nFirstPage)

Attempt to set the number of the first page of this document. Note that setting a negative first page number is legal: this may be useful in printing a portion of the document with offset page numbers from what it would normally print. Note also that not all implementations permit the initial page number to be set, as some implementations simply lack the information as to how this page information should be reflected in the final output.

Argument	Type	Description
nFirstPage	*LONG*	The desired first page number.

Return Value	Meaning
S_OK	The first page was set as requested.
E_FAIL	The first page could not be set to the indicated value.
E_UNEXPECTED	An unknown error occurred.

IPrint::GetPageInfo

HRESULT IPrint::GetPageInfo([out] LONG *nFirstPage, [out] LONG *pcPages)

Return information about the pages in the document.

Argument	Type	Description
pnFirstPage	*LONG**	Location to return the page number of the first page. May be NULL, indicating the caller doesn't need this number. If *IPrint::SetInitialPageNum* has been called, this should contain the same value passed to that method. Otherwise the value is the document's internal first page number.
pcPages	*LONG**	Location to return the total number of pages in this document. May be NULL, indicating the caller doesn't need this number.

Return Value	Meaning
S_OK	Success.
E_UNEXPECTED	An unexpected error occurred.

IPrint::Print

HRESULT IPrint::Print([in] DWORD grfFlags, [in,out] DVTARGETDEVICE **pptd
, [in,out] PAGESET **pppageset, [unique][in,out] STGMEDIUM *pstgmOptions
, [in] IContinueCallback *pcallback, [in] LONG nFirstPage, [out] LONG *pcPagesPrinted
, [out] LONG *pnLastPage)

Print this object on the printer indicated by the DVTARGETDEVICE structure in *ptd*. The DEVMODE in the target device indicates whole-job printer-specific options, such as number of copies, paper size, print quality, etc. It may or may not also contain orientation information in the *dmOrientation* field (this is indicated in the *dmFields* field). If present, then this paper orientation should be used; if absent, then natural orientation as determined by the object content is to be used.

Due to the possibility of user input, the parameters *pptd* and *ppPageSet* are both [in,out] structures. In the absence of user interaction (that is, without PRINTFLAG_PROMPTUSER), both the target device and the page set will necessarily be the same on input and output. However, if the user is prompted for print options, then the object returns target device and page set information appropriate to what the user has actually chosen during interaction.

ppstgmOptions is an [in,out] parameter. On exit, the object should return through **ppstgmOptions* any object-specific information that it would need to reproduce this exact print job. Examples might include whether the user selected "sheet, notes, or both" in a spreadsheet application. The data returned is in the format of a serialized property set. The returned data can usually only be usefully used by passing it back in a subsequent call to the same object; however, that call may have different user interaction flags, different target device, etc. Thus, the caller can cause the exact same document to be printed multiple times in slightly different printing contexts.

Argument	Type	Description
grfFlags	*DWORD*	A bit field whose values are taken from the enumeration PRINTFLAG.
pptd	*DVTARGETDEVICE***	The target device on which the printing is to occur.
ppPageSet	*PAGESET***	Indicates which pages are to be printed.
ppstgmOptions	*STGMEDIUM***	Contains object-specific printing options in the form of a serialized OLE property set. May be NULL in one or both directions.
pCallback	*IContinueCallback**	A callback interface which is to be periodically polled at human-response speeds to determine whether printing should be abandoned. May be NULL.
nFirstPage	*LONG*	The starting page number to print. This overrides any value previously passed to *IPrint::SetInitialPageNum*.
pcPagesPrinted	*LONG**	The place at which the object is to return the actual number of pages that were successfully printed.
pnLastPage	*LONG**	The place at which the object is to return the last legal page number.

Return Value	Meaning
S_OK	Success
PRINT_E_CANCELLED	The print process was canceled. **pcPagesPrinted* indicates the number of pages that were in fact successfully printed before this occurred.
PRINT_E_NOSUCHPAGE	An attempt has been made to print a page which does not exist.
E_UNEXPECTED	An unexpected error occurred.

The IContinueCallback *Interface*

This interface is a generic callback mechanism for interruptible processes that should periodically ask an object with this interface whether to continue the process.

IDL:

```
[
uuid(B722BCCA-4E68-101B-A2BC-00AA00404770)
    , object, pointer_default(unique)
]
interface IContinueCallback : IUnknown
    {
    HRESULT FContinue(void);
    HRESULT FContinuePrinting([in] LONG nCntPrinted
        , [in] LONG nCurPage, [unique][in] wchar_t *pszPrintStatus);
    }
```

The *FContinue* function is a generic continuation request. *FContinuePrinting* carries extra information pertaining to a printing process and is used in the context of *IPrint.*

IContinueCallback::FContinue

HRESULT IContinueCallback::FContinue(void)

Answer as to whether a given generic operation should continue.

Argument	Type	Description
NA	NA	NA

Return Value	Meaning
S_OK	Continue the operation.
S_FALSE	Cancel the operation as soon as possible

IContinueCallback::FContinuePrinting

HRESULT IContinueCallback::FContinuePrinting(cPagesPrinted, nCurrentPage, wszPrintStatus)

Answer as to whether a given lengthy printing operation should continue. Implementations of *IPrint* call back on this method at periodic intervals during the printing process. The *IPrint* implementation should call back at least after printing each page, so that its client can display useful visual feedback to the user. Further, the implementation can legally call back multiple times with the same *cPagesPrinted* and *nCurrentPage* values; this is sometimes useful when a page being printed is complex and it is appropriate to give the user a chance to cancel mid-page.

Argument	Type	Description
cPagesPrinted	*LONG*	The total number of pages printed so far.
nCurrentPage	*LONG*	The page number of the current page being printed.
pszPrintStatus	*LPOLESTR*	Status message about the print job which the recipient of this call may choose to display to the user. May be NULL.

Return Value	Meaning
S_OK	Continue printing.
S_FALSE	Cancel the print job as soon as possible
E_UNEXPECTED	An unknown error occurred.

The IOleCommandTarget *Interface*

The command dispatch interface *IOleCommandTarget* defines a simple and extensible mechanism to query and execute commands which are defined as integer identifiers in a group. The group is identified itself with a GUID. The interface allows a caller to both query for support of commands within a group as well as to instruct the object to execute those commands.

IDL:

```
    [
    uuid(B722BCCB-4E68-101B-A2BC-00AA00404770)
        , object, pointer_default(unique)
    ]
    interface IOleCommandTarget : IUnknown
        {
        typedef [unique] IOleCommandTarget *LPOLECOMMANDTARGET;

        typedef enum
            {
            OLECMDF_SUPPORTED   = 0x00000001,
            OLECMDF_ENABLED     = 0x00000002,
            OLECMDF_LATCHED     = 0x00000004,
            OLECMDF_NINCHED     = 0x00000008
            } OLECMDF;

        typedef struct _tagOLECMD
            {
            ULONG cmdID;
            DWORD cmdf;
            } OLECMD;

        typedef enum
            {
            OLECMDTEXTF_NONE   = 0,
            OLECMDTEXTF_NAME   = 1,
            OLECMDTEXTF_STATUS = 2
            } OLECMDTEXTF;

        typedef struct  _tagOLECMDTEXT
            {
            DWORD cmdtextf;
            ULONG cwActual;
            ULONG cwBuf;
            [size_is(cwBuf)] wchar_t rgwz[];
            } OLECMDTEXT;

        typedef enum
            {
            OLECMDEXECOPT_DODEFAULT      = 0,
            OLECMDEXECOPT_PROMPTUSER     = 1,
            OLECMDEXECOPT_DONTPROMPTUSER = 2,
            OLECMDEXECOPT_SHOWHELP       = 3
            } OLECMDEXECOPT;

        typedef enum
            {
            OLECMDID_OPEN           = 1,
            OLECMDID_NEW            = 2,
            OLECMDID_SAVE           = 3,
            OLECMDID_SAVEAS         = 4,
            OLECMDID_SAVECOPYAS     = 5,
        OLECMDID_PRINT             = 6,
        OLECMDID_PRINTPREVIEW      = 7,
        OLECMDID_PAGESETUP         = 8,
        OLECMDID_SPELL             = 9,
        OLECMDID_PROPERTIES        = 10,
        OLECMDID_CUT               = 11,
        OLECMDID_COPY              = 12,
        OLECMDID_PASTE             = 13,
        OLECMDID_PASTESPECIAL      = 14,
        OLECMDID_UNDO              = 15,
        OLECMDID_REDO              = 16,
        OLECMDID_SELECTALL         = 17,
        OLECMDID_CLEARSELECTION    = 18,
        OLECMDID_ZOOM              = 19,
        OLECMDID_GETZOOMRANGE      = 20
        } OLECMDID;

    [input_sync] HRESULT QueryStatus([unique][in] const GUID *pguidCmdGroup
        , [in] ULONG cCmds, [in,out][size_is(cCmds)] OLECMD *prgCmds
        , [unique][in,out] OLECMDTEXT *pCmdText);
    HRESULT Exec([unique][in] const GUID *pguidCmdGroup
        , [in] DWORD nCmdID, [in] DWORD nCmdExecOpt
        , [unique][in] VARIANTARG *pvaIn
        , [unique][in,out] VARIANTARG *pvaOut);
    };
```

OLECMDF Enumeration

Values from the OLECMDF enumeration are used to fill the value of the *cmdf* field in OLECMD structures as passed to *IOleCommandTarget::QueryStatus*.

Flag	Description
OLECMDF_SUPPORTED	The command is supported by this object.
OLECMDF_ENABLED	The command is available and enabled.
OLECMDF_LATCHED	The command is an on-off toggle and is currently on.
OLECMDF_NINCHED	The command is an on-off toggle but the state cannot be determined because the attribute of this command is found in both on and off states in the relevant selection. This state corresponds to an "indeterminate" state of a 3-state checkbox, for example.

OLECMD Structure

The OLECMD structure is used to associate command flags from the OLECMDF enumeration with a command identifier through *IOleCommandTarget::QueryStatus*.

Field	Type	Description
cmdID	ULONG	A command identifier.
cmdf	DWORD	Flags associated with *cmdID* taken from the OLECMDF enumeration.

OLECMDTEXTF Enumeration

Values from the OLECMDTEXTF enumeration are used to describe what a command target object should store in the OLECMDTEXT structure passed to *IOleCommandTarget::QueryStatus*. One value from this enumeration is stored in the *cmdtextf* of the structure to indicate the desired information.

Flag	Description
OLECMDTEXTF_NONE	No extra information is requested.
OLECMDTEXTF_NAME	The object should return the localized name of the command.
OLECMDTEXTF_STATUS	The object should return a localized status string for the command.

OLECMDTEXT Structure

Used to return a text name or a status string for a single command identifier when used with *IOleCommandTarget::QueryStatus*.

Field	Type	Description
cmdtextf	*DWORD*	Filled on input; a value from the OLECMDTEXTF enumeration describing the information the caller wishes to receive in return.
cwActual	*ULONG*	Filled on output; the number of characters actually written into the *rgwz* buffer before the function returns.
cwBuf	*ULONG*	Filled on input; the size of the string buffer in *cwBuf.*
rgwz	*wchar_t*	A caller allocated array of wide characters to receive the string on output.

OLECMDEXECOPT Enumeration

Flag	Description
OLECMDEXECOPT_PROMPTUSER	Execute the command after taking user input.
OLECMDEXECOPT_DONTPROMPTUSER	Execute the command without prompting the user (for example, clicking on the Print toolbar button, causes the document to be immediately printed without requiring the user input).
OLECMDEXECOPT_DODEFAULT	Caller is not sure whether the user should be prompted or not.
OLECMDEXECOPT_SHOWHELP	Object should show help for the corresponding command and not execute.

OLECMDID Enumeration

See below under "Standard Command List."

IOleCommandTarget::QueryStatus

```
[input_sync] HRESULT QueryStatus([unique][in] const GUID *pguidCmdGroup, [in] ULONG cCmds,
    [in,out][size_is(cCmds)] OLECMD *prgCmds , [unique][in,out] OLECMDTEXT *pCmdText);
```

Queries the object for the status of one or more commands, typically used in WM_INITMENU or WM_INITMENUPOPUP messages, enabling the caller to disable those commands that would be routed to the object but that are not available. The caller passes an array of OLECMD structures in *prgCmds* that describe the commands of interest from the group specified in *pguidCmdGroup*, where each structure's *cmdID* is set to a command identifier and the *cmdf* field is set to zero. The object receiving the call the fills the *cmdf* field for each command with bits taken from the OLECMDF enumeration to describe the status of each command.

The caller can also use this method to get the name or status text of a single command. The called object should first mark the command as described above. If the command is supported (OLECMDF_SUPPORTED) then the object should check the OLECMDTEXTF flags in the OLECMDTEXT structure. If the OLECMDFTEXF_NAME flag is specified, then the object should copy the localized name of the command (for example, "Open", "Copy", etc.) into the *rgwz* field of OLECMDTEXT, paying attention to the size specified by the *cwBuf* field in that same structure.

If, however, the caller specifies OLECMDFTEXTF_STATUS then the object should instead copy a localized status string for the command into the *rgwz* field. The status string is typically contextual, and it depends on the state of the command such as enabled/disabled. If the buffer is not big enough then the object should zero terminate the buffer. Whether the buffer is big enough or not the object must return the total actual size of the string(s), that he attempted to copy, via *cwActual* field.

If the command array contains more than one command, then the textual information should be returned for the first command in the command array that the object supports. Typically this functionality is used to show the status text of a command. Note that the caller can use a stack or global variable for *rgwz*, it not be dynamically allocated memory.

This member function is defined with the [input_sync] attribute, hence the implementing object cannot yield or make another non input_sync RPC call while executing this method.

Argument	Type	Description
pguidCmdGroup	*const GUID **	Unique identifier of the command group which can be NULL to specify the standard group. All the commands that are passed in the *rgCmds* array must belong to this group.
cCmds	*ULONG*	The number of commands in the *prgCmds* array.
prgCmds	*OLECMD **	An caller-allocated array of OLECMD structures where the *cmdID* fields of the structures initialized with the commands being queried.
pcmdText	*OLECMDTEXT **	Pointer to the structure in which to return name and/or status information. Can be NULL to indicate that the caller is not interested in such information.

Return Value	Meaning
S_OK	The command status as any optional strings were returned successfully.
E_POINTER	The *prgCmds* argument is NULL.
E_UNEXPECTED	An unexpected error occurred.
E_FAIL	An error occurred
OLECMDERR_E_UNKNOWNGROUP	*pguidCmdGroup* is non-NULL but does not specify a recognized command group.

Comments:
A command target must implement this function; therefore E_NOTIMPL is not a valid return code.

IOleCommandTarget::Exec

```
HRESULT Exec([unique][in] const GUID *pguidCmdGroup, [in] DWORD nCmdID
    , [in] DWORD nCmdExecOpt, [unique][in] VARIANTARG *pvaIn
    , [unique][in,out] VARIANTARG *pvaOut)
```

Executes a specified command or displays help for a command. As in the case of *IOleCommandTarget::QueryStatus*, the *pguidCmdGroup* and *nCmdID* arguments uniquely identify the command to invoke. The exact action to take is specified in *nCmdExecOpt* (see the OLECMDEXECOPT enumeration for more details).

Most of the commands take no arguments nor do they return any values. Hence, for majority of the commands the caller can pass NULLs for *pvaIn* and *pvaOut*. For the commands which expect one or more input value, the caller can declare and initialize a VARIANTARG variable and pass a pointer to that variable in *pvaIn.*[7] If the input to the command is a single value then the argument can be stored directly in the VARIANTARG and passed to the function. If the command expects multiple arguments then they must be packaged appropriately within the VARIANTARG using one of the supported types (such as *IDispatch, SAFEARRAY*, etc.).

Similarly, if a command returns one or more arguments the caller is expected to declare a VARIANTARG, initialize it to VT_EMPTY, and pass its address in *pvaOut*. If the command returns a single value then the object can store that value directly in *pvaOut*. If the command has multiple output values then it will package those in some way appropriate for the VARIANTARG.

Note that both *pvaIn* and *pvOut* are caller-allocated, thus stack variables are perfectly usable. For commands that take zero or one argument on input and return zero or one values, then no extra memory allocation is necessary.[8] the caller and callee can use stack variables.

The list of *in* and *out* arguments of a command and how they are packaged is unique to each command; such information should be documented with the specification of the command group (see the Zoom command later in this section). In the absence of any specific information the command is assumed to take no arguments and have no return value.

[7] **VARIANTARG is defined in OLE Automation.**

[8] **Most of the types supported by VARIANTARG do not require memory allocation, few of the exceptions are SAFEARRAY and BSTR. For the complete list, see OLE documentation.**

Argument	Type	Description
pguidCmdGroup	*const GUID* *	Unique identifier of the command group which can be NULL to specify the standard group. The command passed in *nCmdID* must belong to this group.
nCmdID	*DWORD*	The command to execute which must be in the group specified with *pguidCmdGroup.*
nCmdExecOpt	*DWORD*	One or more values from the OLECMDEXECOPT enumeration describing how the object should execute the command.
pvaIn	*VARIANTARG* *	Pointer to a VARIANTARG containing input arguments. Can be NULL.
pvaOut	*VARIANTARG* *	Pointer to a VARIANTARG to receive the output return values. Can be NULL.

Return Value	Meaning
S_OK	The command was executed successfully.
E_UNEXPECTED	An unexpected error occurred.
E_FAIL	An error occurred
OLECMDERR_E_UNKNOWNGROUP	*pguidCmdGroup* is non-NULL but does not specify a recognized command group.
OLECMDERR_E_NOTSUPPORTED	The *nCmdID* argument is not recognized as a valid command in the group identified with *pguidCmdGroup.*
OLECMDERR_DISABLED	The command identified with *nCmdID* is currently disabled and cannot be executed.
OLECMDERR_NOHELP	The caller has asked for help on the command identified by *nCmdID* but no help is available.
OLECMDERR_CANCELED	The user canceled the execution of the action.

Comments:
A command target must implement this function; therefore E_NOTIMPL is not a valid return code.

Standard Command List

Following is the list of standard commands that have been defined by Office 95 which are identified as the group with a NULL GUID (that is, *pguidCmdGroup* as passed to *IOleCommandTarget::Exec* is NULL; this is not the same as GUID_NULL, which is *not* used in this context).

Identifier	Description
OLECMDID_OPEN	File Open
OLECMDID_NEW	File New
OLECMDID_SAVE	File Save
OLECMDID_SAVEAS	File Save As
OLECMDID_SAVECOPYAS	File Save Copy As
OLECMDID_PRINT	File Print
OLECMDID_PRINTPREVIEW	File Print Preview
OLECMDID_PAGESETUP	File Page Setup
OLECMDID_SPELL	Tools Spelling
OLECMDID_PROPERTIES	File Properties
OLECMDID_CUT	Edit Cut
OLECMDID_COPY	Edit Copy
OLECMDID_PASTE	Edit Paste
OLECMDID_PASTESPECIAL	Edit Paste Special
OLECMDID_UNDO	Edit Undo
OLECMDID_REDO	Edit Redo
OLECMDID_SELECTALL	Edit Select All
OLECMDID_CLEARSELECTION	Edit Clear
OLECMDID_ZOOM	View Zoom (see below for details)
OLECMDID_GETZOOMRANGE	Retrieves zoom range applicable to View Zoom (see below for details)

The Zoom Commands

Under normal OLE Documents functionality, an object being edited in-place disabled its Zoom control on its toolbar and its

View.Zoom menu are disabled, because logically the Zoom applies to the container document and not the object. With the OLECMDID_ZOOM and OLECMDID_GETZOOMRANGE commands in the standard set for *IOleCommandTarget*, the object now has a means through which it can notify the container's frame object (the one with *IOleInPlaceFrame* as well as *IOleCommandTarget*, if supported) as well as retrieve the zoom range that it should display in its user interface.

OLECMDID_ZOOM

The OLECMDID_ZOOM command takes one LONG argument as input and returns one LONG argument on output. This command is used for three purposes:

- *To query the current zoom value* the caller passes OLECMDEXECOPT_DONTPROMPTUSER as the execute option in *nCmdExecOpt* and NULL for *pvIn*. The object returns the current zoom value in *pvaOut*. When the object goes UI active, it retrieves the current zoom value from the container's frame object using this same mechanism and updates its zoom control with the returned value.
- *To display the Zoom dialog box* the caller passes OLECMDEXECOPT_PROMPTUSER in *nCmdExecOpt*. The caller can optionally pass the initial value for the dialog box through *pvaIn*, otherwise *pvaIn* must be NULL. If the user presses CANCEL, the object returns OLECMDERR_E_CANCELED; if the user presses OK, then the object returns the user selected value in *pvaOut*. When user selects the View.Zoom menu item, the object calls container's frame object in the same manner. The container then zooms the document to the user selected value, and the object updates its Zoom control with that value.
- *To set a Zoom value* the caller passes OLECMDEXECOPT_DONTPROMPTUSER in *nCmdExecOpt* and passes the zoom value to apply through *pvaIn*. The object validates and normalizes the new value and returns the validated value in *pvaOut*. When the user selects a new zoom value (using the Zoom control on the toolbar for instance) the object calls the container's frame object in this manner. The container zooms the document to the normalized value and object updates the Zoom control with that value.

OLECMDID_GETZOOMRANGE

The OLECMDID_GETZOOMRANGE command is used to determine the range of valid zoom values from a command target object. The caller passes MSOCMDEXECOPT_DONTPROMPTUSER in *nCmdExecOpt* and NULL for *pvaIn*. The object returns its zoom range as a DWORD in *pvaOut* where the HIWORD contains the maximum zoom value and the LOWORD contains the minimum zoom value. Typically this command is used when the user drops down the Zoom control on the toolbar of the UI active object. The applications and objects that support this command are required to support all the integral zoom values that are within the (min,max) pair they return.

Appendix: Office Binder Issues

This short appendix describes some of the details concerning Office Binder's implementation of programmatic printing. In addition, one other note is worth mention which is that Binder allows the user to open a document object into a separate window (the semantics of *IOleDocumentView::Open*). It is recommended that Office-compatible DocObjects support separate window activation if possible.

As for printing, Binder uses *IPrint* for Binder level printing, thus DocObjects that wish to work well with Binder must implement *IPrint.* The Binder supports two distinct levels of printing. At the Binder level, users can print multiple sections. At the section level, only the selected section will be printed (implemented via the *IOleCommandTarget* interface).

Binder Level Printing

When printing a section of the Binder level, Binder will be responsible for displaying the user interface elements that are related to print progress, canceling of the print job, and so forth. This will be indicated by the absence of the PRINTFLAG_MAYBOTHERUSER flag in the call to *IPrint::Print.* Binder is always going to call *IPrint::Print* with PRINTFLAG_RECOMPOSETODEVICE bit set. Depending on the user's selection, Binder may set the DM_COLLATE and DM_COPIES bits of *dmFields* field of DVTARGETDEVICE. When DM_COPIES bit is set then the *dmCopies* field contains the number of copies that need to be printed. The document object being printed must look at these fields and use the information they contain when it prints.

When the user selects the *Print to file* option in the print dialog box, then the Binder will call *IPrint::Print* with PRINTFLAG_PRINTTOFILE and it will pass the name of the file (into which the document object must print) through the "*portname*" field of DVTARGETDEVICE. The document object can then put that file name in the DOCINFO structure, and pass it to the WIN32 *StartDoc* API as part of the printing process. This will take handle the "print to file" request.

Section Level Page Setup and Printing

A user can opt to perform Page Setup and Printing at the section level. When Page Setup is chosen, Binder will call the *IOleCommandTarget::Exec* method with OLECMDID_PAGESETUP. This indicates that the object should prompt the user for page-specific options using its Page Setup dialog.

Similarly when printing a section at the section level, the Binder will call the *IOleCommandTarget::Exec* method with OLECMDID_PRINT, indicating that printing is to be performed. The document object should prompt the user with its File/Print dialog and use it's own settings to perform the print job.

During section level printing the object should display any user interface elements that are needed by the user (that is, print job status, cancellation buttons, etc).

Calling *IContinueCallback::FContinuePrinting*

During Binder-level printing, it is important for DocObjects to call *IContinueCallback::FContinuePrinting* often, so that Binder can response quickly if the user presses the Cancel button in the Binder's print dialog box. The document object must call at least once for each page that it is printing. If a specific page will take a long time to compose and print, then the document object should call more often to assure a timely response to the user's commands.

APPENDIX S

IShellView

[Now Supported on Windows NT Beta]

The **IShellView** interface is implemented to present a view in the Windows Explorer or folder windows. The object that exposes **IShellView** is created by a call to **IShellFolder::CreateViewObject**. This provides the channel of communication between a view object and the Explorer's outermost frame window. The communication involves the translation of messages, the state of the frame window (activated or deactivated), and the state of the document window (Activated or deactivated), the merging of menus, and toolbar items.

When to Implement

This interface is implemented by namespace extensions that want to represent themselves in the Explorer's namespace. This object is created by the **IShellFolder** object that hosts the view.

When to Use

These methods are used by the shell view's Explorer window to manipulate objects while they are active.

Methods in Vtable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IOleWindow Methods	Description
GetWindow	Returns a handle to one of the windows participating in in-place activation.
ContextSensitiveHelp	Determines whethar context-sensitive help mode should be entered during an in-place activation session.

IShellView Methods	Description
TranslateAccelerator	Translates accelerator key strokes when a namespace extension's view has the focus.
EnableModeless	Enables or disables modeless dialog boxes. Not in use by the Explorer at this time.
EnableModelessSV	Not in use at this time.
UIActivate	Passas a value when the state of the view window is changed by events not caused by the shell view itself.
Refresh	Responds to user input to refresh the display.
CreateViewWindow	Creates the view window.
DestroyViewWindow	Destroys the view window.
GetCurrentInfo	Returns the folder settings.
AddPropertySheetPages	Allows the view to add pages to tha options property sheet.
SaveViewState	Saves the current view state into a stream obtained by the view by calling **IShellBrowser::GetViewStateStream.**
SelectItem	Changes the state of items within the shell view window.
GetItemObject	Allows callers to get an object that represents something in the view.

IShellView::AddPropertySheetPages

[Now Supported on Windows NT Beta]

Provides a way for the view to add pages to the Options property sheet.

```
HRESULT AddPropertySheetPages(
 DWORD dwReserved,          // Reserved.
 LPFNADDPROPSHEETPAGE lpfn,    // Points to the callback that adds pages
 LPARAM lparam        // lparam to be passed to the callback function
);
```

Parameters

dwReserved
This parameter is reserved for future use.

lpfn
Pointer to the callback function used to add the pages.

lparam
Specifies the *lParam* that must be passed to the callback in the *lpfn* parameter.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

Allows the view to add property pages to the View.Options... property page.

Notes to implementors

The Explorer calls this method when it is opening the View.Options... property sheet. Views can add pages by creating them and calling the callback function with the page handles.

See Also

IShellView, CreatePropertySheetPage

IShellView::CreateViewWindow

[Now Supported on Windows NT Beta]

CreateViewWindow creates a view window. This can be either the right pane of the Explorer or the client window of a folder window.

```
RESULT CreateViewWindow(
 ISHELLLINK *lpPrevView,       // Points to previous view
 LPFOLDERSETTINGS lpfs,    // Points to FOLDERSETTINGS
 IShellBrowser *psb,  // Points to shell browser
 RECT *prcView,         // Points to the rect the defines the view size
 HWND *phWnd           // Points to the returned window handle
);
```

Parameters

lpPrevView
Pointer to the view window being exited. Views can use it to talk to a previous view of the same implementation. This can be used to optimize browsing between like views. This pointer may be NULL.

lpfs
Pointer to a FOLDERSETTINGS structure. The view should use this when creating its view.

psb
Pointer to the current instance of **IShellBrowser**. The view should **AddRef** this pointer and keep it to allow communication with the Explorer window.

prcView
Specifies the dimensions in client coordinates in which the view should create itself.

phWnd
Pointer to the handle of the window being created.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

This is the call that creates the view.

Notes to Callers

Call this method when the view needs to be created.

Notes to Implementors

Create your view window and restore any persistent state by calling **IShellBrowser::GetViewStateStream**.

See Also

IShellView, **IShellBrowser::GetViewStateStream**

IShellView::DestroyViewWindow

[Now Supported on Windows NT Beta]

DestroyViewWindow destroys the view window.

HRESULT DestroyViewWindow(

Parameters

This method has no parameters.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

The Explorer calls this method when a folder window or the Explorer is being closed.

Notes to Implementors

Clean up all state that represents the view, including the window and any other associated resources.

See Also

IShellView

IShellView::EnableModeless

[Now Supported on Windows NT Beta]

If the view owns any modeless dialog boxes, it should disable all of them when this member is called with FALSE and keep them disabled until it is called again with TRUE.

```
HRESULT EnableModeless(
 LPFOLDERSETTINGS fEnable        // Boolean flags
);
```

Parameters

fEnable
Specifies TRUE to enable modeless dialog box windows, FALSE to disable them.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

See Also

IShellView

IShellView::GetCurrentInfo

[Now Supported on Windows NT Beta]

Obtains information about the current folder settings.

```
HRESULT GetCurrentInfo(
  LPFOLDERSETTINGS lpfs     // Points to the folder settings
);
```

Parameters

lpfs
Pointer to a FOLDERSETTINGS structure to receive the settings.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

The Explorer uses **GetCurrentInfo** to query the view for standard settings.

Notes to Callers

Used to get the current view settings of the view.

Notes to Implementors

Return as many of the settings as apply. This is intended to let browsing from view to view maintain the same basic settings. For example, if the user sets Details view, going from one folder to the other in Explorer mode, it should remain in Details view.

See Also

IShellView

IShellView::GetItemObject

[Now Supported on Windows NT Beta]

Returns an interface that refers to data presented in the view.

```
HRESULT GetItemObject(
  UINT uItem,   // Specifies background object constants
  REFIIDriid,   // Identifies the interface to return
  LPVOID *ppv          // Address that receives the interface pointer
);
```

Parameters

uItem
Specifies constants that refer to an aspect of the view. It can be any of the following values.

Value Meaning
SVGIO_BACKGROUND Refers to the background of the view. It is used with IID_IContextMenu to get a context menu for the view background.
SVGIO_SELECTION Refers to the currently selected items. IID_IDataObject uses this constant to get a data object that represents the selected items.
SVGIO_ALLVIEW Same as SVGIO_SELECTION but refers to all items in the view.

riid
Identifier of the interface to return.

ppv
Address that receives the interface pointer. If an error occurs, the pointer returned must be NULL.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

Used by the common dialogs to get the selected items from the view.

See Also

IShellView

IShellView::Refresh

[Now Supported on Windows NT Beta]

Refreshes the view's contents in response to an event such as when a user hits the F5 key.

HRESULT Refresh(

Parameters

This method has no parameters.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

Tells the view to refresh its contents, revalidating any view information it has.

Notes to Callers

The Explorer calls this method when F5 is pressed on an already open view.

Notes to Implementors

Refill the view by going to any underlying storage for the contents.

See Also

IShellView

IShellView::SaveViewState

[Now Supported on Windows NT Beta]

Allows the shell view to store its view settings so the current state can be restored during a subsequent browsing session.

HRESULT SaveViewState(

Parameters

This method has no parameters.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

The shell view is supposed to get a view stream by calling **IShellBrowser::GetViewStateStream** and store the current view state in that stream.

Notes to Callers

The Explorer calls this method when it wants to save the view state for a view.

Notes to Implementors

Be sure to make the format of the data stored in the stream robust and versionable.

See Also

IShellBrowser::GetViewStateStream, **IShellView**

IShellView::SelectItem

[Now Supported on Windows NT Beta]

Changes the selection state of one or more items within the shell view window.

```
HRESULT SelectItem(
 LPCITEMIDLIST pidlItem,      // Points to item ID list
 UINT uFlags  // Specifies the selection state
);
```

Parameters

pidlItem
Pointer to the item ID list. If this parameter is NULL and *uFlags* is SVSI_DESELECTOTHERS, all items should be deselected.

uFlags
Flag specifying what type of selection to apply. This parameter can be one of the following values:

Value	Meaning
SVSI_DESELECT	Deselect the specified item.
SVSI_DESELECTOTHERS	If *pidlItem* is NULL, deselect all items.
SVSI_EDIT	Put the *pidlItem* in edit mode.
SVSI_ENSUREVISIBLE	Ensure the item is displayed on the screen.
SVSI_FOCUSED	The item should be given the focus.
SVSI_SELECT	The item should be selected.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

This method is used to implement functionality in the Explorer.

Notes to Implementors

SelectItem is used to implement the File Target command of the shell shortcut property sheet.

See Also

IShellView

IShellView::TranslateAccelerator

[Now Supported on Windows NT Beta]

Processes menu accelerator-key messages from the container's message queue.

```
HRESULT TranslateAccelerator(
 LPMSG lpmsg          // Points to a message that may need translating.
);
```

Parameters

lpmsg
Pointer to the message that might need to be translated.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Returning S_OK indicates that the message was translated and should not be translated or dispatched by the Explorer.

Remarks

TranslateAccelerator is called by the Explorer to let the view translate its accelerators.

Notes to Callers

The Explorer calls this method before any other translation if the view has the focus. If the view does not have the focus (if the tree has it, for example) this is called after the Explorer translates its own accelerators.

Notes to Implementors

By default, the view should return S_FALSE so that the Explorer can either do it's own accelerator translation or normal menu dispatching. The view should return S_OK only if it has processed the message as the accelerator and does not want the Explorer to process it further.

See Also

IShellView

IShellView::UIActivate

[Now Supported on Windows NT Beta]

Called by the Explorer whenever the activation state of the view window is changed by a certain event that is not caused by the shell view itself. For example, if the TAB key is pressed when the tree has the focus, the view should be given the focus.

```
HRESULT UIActivate(
 UINT uState  // activation state flag
);
```

Parameters

uState
Flag specifying thę activation state of the window. This parameter can be one of the following values:

Value   Meaning
SVUIA_ACTIVATE_FOCUS   The Explorer has just created the view window with the input focus. This means the shell view should be able to set menu items appropriate for the focused state.
SVUIA_ACTIVATE_NOFOCUS   The shell view is either losing the input focus or it has just been created without the input focus. The shell view should be able to set menu items appropriate for the nonfocused state. This means no selection-specific items should be added.
SVUIA_DEACTIVATE   The Explorer is about to destroy the shell view window. The shell view should remove all extended UIs, typically merged menu and modeless popup windows.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

To remerge menu items, the shell view typically hooks the WM_SETFOCUS message and calls **IShellBrowser::OnViewWindowActivated** before remerging. The shell view should not hook the WM_KILLFOCUS message to remerge menu items.

Notes to Callers

Call this method to inform the view of activation state change.

Notes to Implementors

Use this method to track activation state and change any behavior, as appropriate.

See Also

IShellView

IShellBrowser

[Now Supported on Windows NT Beta]

The **IShellBrowser** interface provides services for namespace extensions and is the companion to the **IShellView** interface implemented by namespace extensions.. It is similar to the "site" interfaces that are often found in OLE hosting scenarios, such as **IOleControl** and **IOleControlSite**. This allows the extension to communicate with the host of the namespace, providing UI elements like menus, status text, and tool bars. This interface also provides the extension with a way to access storage to save its persistent view state.

IShellBrowser derives from **IOleWindow** and it represents the container's top-level window, allowing the contained views to insert their menus into the composite menu, install the composite menu into the appropriate window frame and remove the container's menu elements from the composite menu. It sets and displays status text relevant to the in-place object. It also enables or disables the frame's modeless dialog boxes, and translates accelerator keystrokes intended for the container's frame.

When to Implement

You do not implement this interface directly. **IShellBrowser** is implemented by the Windows Explorer and by the Windows File Open Dialog.

When to Use

When implementing a namespace extension, notably **IShellView**, you will use the **IShellBrowser** implementation that is passed to use **IShellBrowser::CreateViewWindow** to communicate with the Explorer.

Methods in Vtable Order

IUnknown Methods	**Description**
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IOleWindow Methods	**Description**
GetWindow	Returns a handle to one of the windows participating in in-place activation.
ContextSensitiveHelp	Determines whether context-sensitive help mode should be entered during an in-place activation session.

IShellBrowser Methods	**Description**
InsertMenusSB	Inserts the Explorer's menu items to an empty menu craated by the view.
SetMenuSB	Installs the composite menu in the Explorer.
RemoveMenusSB	Gives the container a chance to remove its items from a composite menu. It perform tasks that are the opposite of **InsertMenuSB**.
SetStatusTextSB	Sets and displays status text in the Explorer window.
EnableModelessSB	Enables or disables modeless windows of the Explorer, such as a floating toolbar.
TranslateAcceleratorSB	Reserved for future use.
BrowseObject	Tells the Explorer to browse in another folder.
GetViewStateStream	Returns a view-specific stream that can be used to read and write the persistent data for a view.
GetControlWindow	Gets the window handle of an Explorer control.
SendControlMsg	Sends messages to Explorer controls.
QueryActiveShellView	Returns the currently activated (displayed) shellview object.
OnViewWindowActive	Informs the Explorer that the view was activated.
SetToolbarItems	Adds toolbar items to the Explorer's toolbar.

IShellBrowser::BrowseObject

[Now Supported on Windows NT Beta]

Tells the Explorer to browse to another folder.

HRESULT BrowseObject(
 LPCITEMIDLIST *pidl*, // Address of item identifier list
 UINT **wFlags*   // Specifies the folder to be browsed
);

Parameters

pidl

Address of an ITEMIDLIST (item identifier list) structure that specifies an object's location. This value is dependent on the *wFlags* parameter.

wFlags

Flag specifying the folder to be browsed. It can be zero or more of the following values. The first three specify whether another window is to be created.

SBSP_SAMEBROWSER   Browse to another folder with the same Explorer window.
SBSP_NEWBROWSER   Creates another window for the specified folder.
SBSP_DEFBROWSER The default behavior is to respect the view option (the user setting to create new windows or to browse in place). In most cases, callers should use this flag.

The following flags specify either the open, explore, or default mode. These values are ignored if SBSP_SAMEBROWSER or (SBSP_DEFBROWSER && (single window browser || explorer)).

SBSP_OPENMODE   Use a normal folder window.
SBSP_EXPLOREMODE   Use an Explorer window.
SBSP_DEFMODE   Us the same one as the current window.

The following flags specify the *pidl* parameter's category:

SBSP_ABSOLUTE   An absolute pidl (relative from the desktop).
SBSP_RELATIVE   A relative pidl (relative from the current folder).
SBSP_PARENT   Browse the parent folder (ignores the pidl).

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

Views can use this method to force the Explorer to browse to a specific place in the namespace. Typically, these are folders contained in the view.

See Also

IShellBrowser

IShellBrowser::EnableModelessSB

[Now Supported on Windows NT Beta]

Tells the Explorer to enable or disable its modeless dialog boxes.

```
HRESULT EnableModelessSB(
 BOOL fEnable          // Enable or disable modeless dialog
);
```

Parameters

fEnable
Specifies whether the modeless dialog boxes are to be enabled by specifying TRUE or disabled by specifying FALSE.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

This method is similar to **IOleInPlaceFrame::EnableModeless**. Although the current version of the Explorer does not have any modeless dialog boxes, the view should call this member appropriately when it wants to disable or enable modeless dialog boxes associated with the Explorer window.

See Also

IShellBrowser

IShellBrowser::GetControlWindow

[Now Supported on Windows NT Beta]

GetControlWindow can be called by the shell view object to get the window handle of an Explorer control, either for a toolbar or for a status window.

```
HRESULT GetControlWindow(
 UINT id,        // Identifier of an Explorer control
 HWND *lphwnd         // Handle of the control's window
);
```

Parameters

id
  Specifies the identifer for either a toolbar (FCW_TOOLBAR), for a status window (FCW_STATUS), or for a tree (FCW_TREE).

lphwnd
  Pointer to the window handle of the Explorer control.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

GetControlWindow is used so views can directly manipulate the toolbar and status bar. FCW_TREE should be used only to sense whether the tree is present; that is, whether the folder is in Explorer mode or folder mode.

Notes to Callers

This is used to manipulate and test the state of these windows. Do not send messages directly to these controls; instead, use **IShellBrowser::SendControlMsg**. Be prepared for the returns of this call to be NULL. Future versions of the Explorer may not include a toolbar, status bar, or tree window.

Notes to Implementors

IShellBrowser::GetControlWindow returns the *hwnds* of these controls if they exist in your implementation.

See Also

IShellBrowser

IShellBrowser::GetViewStateStream

[Now Supported on Windows NT Beta]

The browser provides an **IStream** interface as the storage for view-specific state information.

```
HRESULT GetViewStateStream(
 DWORD grfMode,       // Specifies the mode
 LPSTREAM *ppStrm  // Points to the LPSTREAM variable
);
```

Parameters

grfMode
Specifiies the read-write access. This may be set to STGM_READ, STGM_WRITE, or STGM_READWRITE. For more information about these values see the STGM enumeration.

ppStrm
Pointer to the address of the LPSTREAM variable to be filled.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

Used to save and restore the persistent state for a view. For example, the icon positions the column widths, and the current scroll position.

Notes to Callers:

Use **GetViewStateStream** when the view is being created to read in the saved view state and when the view is being closed to save any changes to the view state. Typically, the view calls this member with STGM_READ when creating a view window and with STGM_WRITE when the **SaveViewState** method of its **IShellView** interface is called.

Notes to Implementors:

Each shell view should have its own view stream. The Explorer implements an MRU (most recently used) list of view streams that are stored on a per-user basis in the registry.

See Also

IShellBrowser

IShellBrowser::InsertMenusSB

[Now Supported on Windows NT Beta]

Allows the Explorer to insert its menu groups into the composite menu being displayed while viewing or using an extended namespace.

```
HRESULT InsertMenusSB(
 HMENU hmenuShared,          // A handle to an empty menu
 LPOLEMENUGROUPWIDTHS lpMenuWidths // Points to OLEMENUGROUPWIDTHS
);
```

Parameters

hmenuShared
  Specifies a handle to an empty menu.

lpMenuWidths
  Points to an OLEMENUGROUPWIDTHS array of 6 LONG values. The container fills in elements 0,2, and 4 to reflect the number of menu elements it provided in the File, View, and Window menu groups.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

This method is similar to **IOleInPlaceFrame::InsertMenus**. The Explorer puts File and Edit pulldown menus in the File menu group, View and Tools in the Container menu group, and Help in the Window menu group. Each pulldown menu will have a unique identifier, FCIDM_MENU_FILE/EDIT/VIEW/TOOLS/HELP. The view is allowed to insert menu items into those submenus by their identifiers, which is different from OLE's in-place activation mechanism. The command IDs for menus that the view inserts into either the Explorer's submenus or its own submenus, must be between FCIDM_SHVIEWFIRST and FCIDM_SHVIEWLAST.

Notes to Callers

This method is called by namespace extensions when they are first being activated so they can insert their menus into the frame-level user interface.

The object application asks the container to add its menus to the menu specified in *hmenuShared* and to set the group counts in the OLEMENUGROUPWIDTHS array pointed to by *lpMenuWidths*. The object application then adds its own menus and counts. Objects can call **IOleInPlaceFrame::InsertMenus** as many times as necessary to build up the composite menus. The container should use the initial menu handle associated with the composite menu for all items in the drop-down menus.

Notes to Implementors

For **IShellBrowser** implementations, the menu identifiers must be in the range of FCIDM_BROWSERFIRST to FCIDM_BROWSERLAST.

See Also

IShellBrowser

IShellBrowser::OnViewWindowActive

[Now Supported on Windows NT Beta]

The shell view window calls **OnViewWindowActive** when the view window or one of its child windows gets the focus.

```
HRESULT OnViewWindowActive(
 IShellView *ppshv     // Points to the view's address
);
```

Parameters

ppshv
Points to the address of the currently active shell view object.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

The view must pass its **IShellView** implementation to this routine, although the current version of the Explorer does not use this parameter.

Notes to Callers

The shell view window must call this member before calling **IShellBrowser::InsertMenusSB** because it will insert a different set of menu items depending on whether the view has the focus.

Notes to Implementors

Lets you know that the view is getting the focus, for example, on a mouse click.

See Also

IShellBrowser

IShellBrowser::QueryActiveShellView

[Now Supported on Windows NT Beta]

QueryActiveShellView returns the currently activated (displayed) shell view object.

```
HRESULT QueryActiveShellView(
 IShellView **ppshv    // Points to the view's address
);
```

Parameters

ppshv
Points to the address of the currently active shell view object.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Notes to callers

QueryActiveShellView is useful because it is possible for an **IShellBrowser** to host several shell views simultaneously. However, the current version of the Explorer does not do this.

See Also

IShellBrowser

IShellBrowser::RemoveMenusSB

[Now Supported on Windows NT Beta]

Gives the container a chance to remove its menu elements from the in-place composite menu and free all associated resources.

```
HRESULT RemoveMenusSB(
 HMENU hmenuShared          // Handle to in-place composite menu
);
```

Parameters

hmenuShared
Specifies a handle to the in-place composite menu that was constructed by calls to **IShellBrowser::InsertMenusSB** and the Win32 **InsertMenu** function.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

This method is similar to **IOleInPlaceFrame::RemoveMenus**.

The object should always give the container a chance to remove its menu elements from the composite menu before deactivating the shared user interface.

Notes to Callers

Called by the object application while it is being UI-deactivated to remove its menus.

See Also

IShellBrowser

IShellBrowser::SendControlMsg

[Now Supported on Windows NT Beta]

SendControlMsg can be called by the shell view object to send control messages to an Explorer control, either for a toobar or for a status bar window.

```
HRESULT SendControlMsg(
 UINT id,        // Identifies a control
 UINT uMsg,  // Specifies the message to be sent
 WPARAM wParam,    // Depends on uMsg
 LPARAM lParam,      // Depends on uMsg
 LRESULT *pref        // Points to the SendMessage return value
);
```

Parameters

id
Specifies the identifer for either a toolbar (FCW_TOOLBAR) or for a status bar window (FCW_STATUS).

uMsg
Specifies the message to be sent to the control.

wParam
This value depends on the message specified in the *uMsg* parameter.

lParam
This value depends on the message specified in the *uMsg* parameter.

pret
Pointer to the return value of the **SendMessage** function.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

Refer to the commctrl.h header file to find the messages that can be sent to the toolbar or status bar control.

Notes to Callers

Use of this call requires diligent attention because leaving either the status bar or toolbar in an inappropriate state will affect the performance of the Explorer.

Notes to Implementors

If your Explorer does not have these controls you can return E_NOTIMPL.

See Also

IShellBrowser

IShellBrowser::SetMenuSB

[Now Supported on Windows NT Beta]

Installs the composite menu in the view window. Similar to **IOleInPlaceFrame::SetMenu**.

```
RESULT SetMenuSB(
 HMENU hmenuShared,          // A handle to the composite menu
 HOLEMENU holemenuReserved      // Reserved for future use
);
```

Parameters

hmenuShared
Specifiies a handle to the composite menu constructed by calls to **IShellBrowser::InsertMenusSB** and the Win32 **InsertMenu** function.

holemenuReserved
Reserved for future use.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

This method is similar to **IOleInPlaceFrame::SetMenu**. However, the Explorer performs menu dispatch based on the menu item ID.

The availability of specific menu items depends on whether the view has the focus. Accordingly, it is necessary to call **IShellBrowser::OnViewWindowActive** whenever the view window (or one of it's child windows) has the focus.

Notes to Callers

The object calls **IShellBrowser::SetMenuSB** to ask the container to install the composite menu structure set up by calls to **IShellBrowser::InsertMenusSB**.

Notes to Implementers

A container's implementation of this method should call the Windows **SetMenu** function.

See Also

IShellBrowser

IShellBrowser::SetStatusTextSB

[Now Supported on Windows NT Beta]

Sets and displays status text about the in-place object in the container's frame-window status line.

```
HRESULT SetStatusTextSB(
 LPCOLESTR lpszStatusText  // Address of string with the message
);
```

Parameters

lpszStatusText
Points to a null-terminated character string containing the message to display.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

It is also possible to send messages directly to the status window by using **SendControlMsg**.

Notes to Callers

Use this method to set the contents of the status bar.

See Also

IShellBrowser

IShellBrowser::SetToolbarItems

[Now Supported on Windows NT Beta]

The **SetToolbarItems** method can be called by the view to add toolbar items to the Explorer's toolbar.

```
HRESULT SetToolbarItems(
 LPTBBUTTON lpButtons,      // Points to an array of items
 UINT nButtons,          // Number of buttons in the array
 UINT uFlags  // Specifies button location
);
```

Parameters

lpButtons
Points to an array of toolbar items.

nButtons
Number of buttons in the *lpButtons* array.

uFlags
Flags specifying where the toolbar buttons should go.

FCT_ADDTOEND Add at the right side of the toolbar.
FCT_CONFIGABLE Not implemented.
FCT_MERGE Merge the toolbar items instead of replacing all of the buttons with those provided by the view. This is the recommended choice.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

This is the way toolbars are merged into the Explorer's toolbar.

Notes to Callers

See the Common Controls TOOLBAR control for the definition of TBBUTTON.

See Also

IShellBrowser

IShellBrowser::TranslateAcceleratorSB

[Now Supported on Windows NT Beta]

This method is not used in the Explorer at this time.

```
HRESULT TranslateAcceleratorSB(
 LPMSG lpmsg,          // Points to an MSG structure
 WORD wID   // Contains the command identifier value
);
```

Parameters

lpmsg
   Points to an **MSG** structure containing the keystroke message.

wID
   Contains the command identifier value corresponding to the keystroke in the container-provided accelerator table. Containers should use this value instead of translating again.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

This method is similar to **IOleInPlaceFrame::TranslateAccelerator** but is not used.

See Also

IShellBrowser

APPENDIX S - Page 26

IShellFolder

[Now Supported on Windows NT Beta]

The **IShellFolder** interface is used to manage folders.

When to Implement

Implement **IShellFolder** for objects that extend the shell's namespace. For example, if you create a separate name space that requires a rooted Explorer; or if you install a new name space directly within the hierarchy of the system name space.Only you know anything about the contents of your name space so you are responsible for implementing everything needed to access your data.

When to Use

Use **IShellFolder** when you need to display or operate on the contents of the shell's namespace. Objects that support **IShellFolder** are usually created by other shell folder objects, with the root object (the Desktop shell folder) being returned from the **SHGetDesktopFolder** function.

Methods in Vtable Order

IUnknown Methods	Description
QueryInterface	Returns pointers to supported interfaces.
AddRef	Increments reference count.
Release	Decrements reference count.

IShellFolder Methods	Description
ParseDisplayName	Translates a display name into an item identifier list.
EnumObjects	Enumerates the objects in a folder.
BindToObject	Retrieves the **IShellFolder** interface for the specified subfolder.
BindToStorage	Returns the storage instance of a subfolder.
CompareIDs	Compares two item identifier lists and returns the result.
CreateViewObject	Creates a view object of the folder itself.
GetAttributesOf	Retrieves the attributes of the specified file object or subfolder.
GetUIObjectOf	Creates an OLE interface that can be used to carry out operations on a file object or subfolder.
GetDisplayNameOf	Retreives the display name of a file object or subfolder.
SetNameOf	Sets the display name of the specified file object or subfolder and changes its identifier accordingly.

Shell's Namespace

A *namespace* is a collection of symbols, such as database keys or file and directory names.

Shell's Namespace Interfaces

The following interfaces are used with the shell's namespace.

IShellFolder
IEnumIDList

IShellFolder::BindToObject

[Now Supported on Windows NT Beta]

Creates an **IShellFolder** object for a subfolder.

```
HRESULT BindToObject(
 LPCITEMIDLIST pidl, // Pointer to an ITEMIDLIST
 LPBC pbcReserved, // Reserved___specify NULL
 REFIID riid,    // Interface to return
 LPVOID *ppvOut       // Address that receives interface pointer
);
```

Parameters

pidl
Pointer to an ITEMIDLIST structure that identifies the subfolder relative to its parent folder.

pbcReserved
Reserved. Callers should specify NULL for this parameter; callees should ignore it.

riid
Identifier of the interface to return. This parameter must point to the IID_IShellFolder interface identifier.

ppvOut
Address that receives the interface pointer. If an error occurs, a NULL pointer is returned in this address.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

Use **BindToObject** to access the COM interface to the sub-folder or sub-object.

See Also

ITEMIDLIST, SHGetDeskTopFolder

IShellFolder::BindToStorage

[Now Supported on Windows NT Beta]

Reserved for a future use. This method should return E_NOTIMPL.

APPENDIX S - Page 29

IShellFolder::CompareIDs

[Now Supported on Windows NT Beta]

Determines the relative ordering of two file objects or folders, given their item identifier lists.

```
HRESULT CompareIDs(
 LPARAM lParam,        // Type of comparison to perform
 LPCITEMIDLIST pidl1,        // Address of ITEMIDLIST structure
 LPCITEMIDLIST pidl2         // Address of ITEMIDLIST structure
);
```

Parameters

lParam
Value specifying the type of comparison to perform. The calling application should always specify zero, indicating that the two items should be sorted by name.

pidl1 and *pidl2*
Addresses of two **ITEMIDLIST** structures that uniquely identify the items to be compared. Both item identifier lists are relative to the parent folder.

Return Values

Returns a handle to a result code. If this method is successful, the CODE field of the status code (SCODE) has the following meaning:

CODE field	Meaning
Less than zero	The first item should precede the second (*pidl1* < *pidl2*).
Greater than zero	The first item should follow the second (*pidl1* > *pidl2*)
Zero	The two items are the same (*pidl1* = *pidl2*).

Remarks

Passing 0 as the *lParam* indicates sort by name. 0x00000001-0x7fffffff are for folder specific sorting rules. 0x80000000-0xfffffff are used the system.

See Also

ITEMIDLIST

IShellFolder::CreateViewObject

[Now Supported on Windows NT Beta]

Creates a view object of a folder.

```
HRESULT CreateViewObject(
 HWND hwndOwner,  // Handle of owner window
 REFIID riid,  // Interface identifier
 LPVOID *ppvOut       // Reserved
);
```

Parameters

hwndOwner

Specifies the owner window for any modal dialog boxes or message boxes within this call. It may be different from *hwndParen* passed in a call to **IShellView::CreateViewWindow**.

Handle of the owner window from which to create the view object.

riid

Identifier of the interface to return.

ppvOut

Specifies the address that receives a pointer to the view object.

Return Values

Returns NOERROR if successful or an OLE defined error value otherwiise.

Remarks

It is important to remember that the COM object created by **CreateViewObject** must be a different object than the shell folder object. The Explorer may call **CreateViewObject** more than once to create more than one view object and expects them to behave as independent objects. A new view object must be created for each call.

IShellFolder::EnumObjects

[Now Supported on Windows NT Beta]

Determines the contents of a folder by creating an item enumeration object (a set of item identifiers) that can be retrieved using the **IEnumIDList** interface.

```
HRESULT EnumObjects(
 HWND hwndOwner,   // Handle of owner window
 DWORD grfFlags,      // ems to include in enumeration
 LPENUMIDLIST *ppenumIDList         // Pointer to IEnumIDList
);
```

Parameters

hwndOwner
Handle of the owner window that the client should specify if it displays a dialog box or message box.

grfFlags
Flags determining which items to iclude in the enumeration. For a list of possible values, see the description of the **SHCONTF** type.

ppenumIDList
Address that receives a pointer to the **IEnumIDList** interface created by this method. If an error occurs, a NULL pointer is returned in this address.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

The calling application must free the returned **IEnumIDList** object by calling its **Release** method.

This method is similar to the method defined by OLE.

See Also

IEnumIDList, **IOleContainer::EnumObjects**, **SHGetDeskTopFolder**

IShellFolder::GetAttributesOf

[Now Supported on Windows NT Beta]

Retrieves the attributes of one or more file objects or subfolders.

```
HRESULT GetAttributesOf(
 UINT cidl,      // Number of file objects
 LPCITEMIDLIST *apidl,       // Pointer to array of pointers to ITEMIDLIST structures
 ULONG *rgfInOut     // Address of value containing attributes of the file objects
);
```

Parameters

cidl
Number of file objects to get the attributes of.

apidl
Pointer to an array of pointers to **ITEMIDLIST** structures, each of which uniquely identifies a file object relative to the parent folder. Each **ITEMIDLIST** structure must contain exactly one **SHITEMID** structure followed by a terminating zero.

rgfInOut
Address of a ULONG value that specifies the common (logically AND'ed) attributes of specified file objects.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

The following tables list the attribute flags that may be returned by this method. File object attributes include capability flags, display attributes, contents flags, and miscellaneous attributes.

A file object's capability flags may include zero or more of the following values:

SFGAO_CANCOPY   The specified file objects or folders can be copied (same value as the DROPEFFECT_COPY flag).
SFGAO_CANDELETE The specified file objects or folders can be deleted.
SFGAO_CANLINK   It is possible to create shortcuts for the specified file objects or folders (same value as the DROPEFFECT_LINK flag).
SFGAO_CANMOVE   The specified file objects or folders can be moved (same value as the DROPEFFECT_MOVE flag).
SFGAO_CANRENAME The specified file objects or folders can be renamed.
SFGAO_CAPABILITYMASK   Mask for the capability flags.
SFGAO_DROPTARGET   The specified file objects or folders are drop targets.
SFGAO_HASPROPSHEET   The specified file objects or folders have property sheets.

A file object's display attributes may include zero or more of the following values:

SFGAO_DISPLAYATTRMASK Mask for the display attributes.
SFGAO_GHOSTED   The specified file objects or folders should be displayed using a ghosted icon.
SFGAO_LINK The specified file objects are shortcuts.
SFGAO_READONLY   The specified file objects or folders are read-only.
SFGAO_SHARE   The specified folders are shared.

A file object's contents flags may include zero or more of the following values:

SFGAO_CONTENTSMASK   Mask for the contents attributes.
SFGAO_HASSUBFOLDER   The specified folders have subfolders (and are, therefore, expandable in the left pane of Windows Explorer).

A file object may have zero or more of the following miscellaneous attributes:

SFGAO_FILESYSTEM The specified folders or file objects are part of the file system (that is, they are files, directories, or root directories).
SFGAO_FILESYSANCESTOR The specified folders contain one or more file system folders.
SFGAO_FOLDER The specified items are folders.
SFGAO_REMOVABLE The specified file objects or folders are on removable media.
SFGAO_VALIDATE Validate cached information.

Remarks

You can optimize this operation by not returning unspecified flags.

See Also

ITEMIDLIST, SHITEMID

IShellFolder::GetDisplayNameOf

[Now Supported on Windows NT Beta]

Retrieves the display name for the specified file object or subfolder, returning it in a **STRRET** structure.

```
HRESULT GetDisplayNameOf(
 LPCITEMIDLIST pidl, // Pointer to an ITEMIDLIST
 DWORD uFlags,       // Type of display to return
 LPSTRRET lpName     // Pointer to a STRRET structure
);
```

Parameters

pidl
Pointer to an ITEMIDLIST structure that uniquely identifies the file object or subfolder relative to the parent folder.

uFlags
Value indicating the type of display name to return. For a list of possible values, see the description of the **SHGNO** enumerated type.

lpName
Pointer to a **STRRET** structure in which to return the display name. The string returned in this structure depends on the type of display name requested.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

If the ID contains the display name (in the local character set), it returns the offset to the name. If not, it returns a pointer to the display name string (UNICODE) allocated by the task allocator, or it fills in a buffer. The type of string returned depends on the type of display specified. Values identifying different types of display names are contained in the enumeration SHGNO.

See Also

ITEMIDLIST, STRRET, SHGNO

IShellFolder::GetUIObjectOf

[Now Supported on Windows NT Beta]

Creates a COM object that can be used to carry out actions on the specified file objects or folders, typically, to create context menus or carry out drag-and-drop operations.

```
HRESULT GetUIObjectOf(
 HWND hwndOwner,  // Handle to owner window
 UINT cidl,       // Number of objects specified in apidl
 LPCITEMIDLIST *apidl,          // Pointer to an array of pointers to an ITEMIDLIST structure
 REFIID riid,  // Interface to return
 UINT *prgflnOut,        // Reserved
 LPVOID *ppvOut          // Address to receive interface pointer
);
```

Parameters

hwndOwner
Handle of the owner window that the client should specify if it displays a dialog box or message box.

cidl
Number of file objects or subfolders specified by *apidl*.

apidl
Pointer to an array of pointers to **ITEMIDLIST** structures, each of which uniquely identifies a file object or subfolder relative to the parent folder. Each item identifier list must contain exactly one **SHITEMID** structure followed by a terminating zero.

riid
Specifies the type and the interface of the COM object to return. This parameter can be a pointer to the IID_IExtractIcon, IID_IContextMenu, IID_IDataObject, or IID_IDropTarget interface identifier.

prgflnOut
Reserved.

ppvOut
Address that receives the interface pointer. If an error occurs, a NULL pointer is returned in this address.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

GetUIObjectOf creates a UI object to be used for specified objects. Either IID_IDataObject (transfer operations) or IID_IContextMenu (context menu operations) is passed in the *riid* parameter.

See Also

ITEMIDLIST, **SHITEMID**

IShellFolder::ParseDisplayName

[Now Supported on Windows NT Beta]

Translates a file object or folder's display name into an item identifier.

```
HRESULT ParseDisplayName(
 HWND hwndOwner,  // Handle of owner window
 LPBC pbcReserved,  // Reserved
 LPOLESTR lpszDisplayName,      // Pointer to diplay name
 ULONG *pchEaten,  // Pointer to value for parsed characters
 LPITEMIDLIST *ppidl,      // Pointer to new item identifier list
 ULONG *pdwAttributes      // Address receiving attributes of file object
);
```

Parameters

hwndOwner
Handle of the owner window that the client should specify if it displays a dialog box or message box.

pbcReserved
Reserved; this parameter is always NULL.

lpszDisplayName
Pointer to a null-terminated Unicode string specifying the display name. This parameter must be a display name for parsing ___ that is, a display name retrieved using the SHGDN_FORPARSING value.

pchEaten
Pointer to an unsigned long value that receives the number of characters of the display name that were parsed.

ppidl
Address that receives a pointer to the new item identifier list for the object. If an error occurs, a NULL is returned in this address.

The returned item identifier list specifies the relative path (from the parent folder) that corresponds to the specified display name. It contains only one **SHITEMID** structure followed by a terminating zero.

pdwAttributes
Address that receives the attributes of the file object. Can be NULL if the caller does not need attribute data.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

This method is similar to the **IParseDisplayName::ParseDisplayName** method defined by OLE.

See Also

IParseDisplayName::ParseDisplayName, **IShellLink**, **SHITEMID**

IShellFolder::SetNameOf

[Now Supported on Windows NT Beta]

Changes the name of a file object or subfolder, changing its item identifier in the process.

```
HRESULT SetNameOf(
 HWND hwndOwner,  // Handle of owner window
 LPCITEMIDLIST pidl, // Pointer to an ITEMIDLIST structure
 LPCOLESTR lpszName,        // Pointer to string specifying new display name
 DWORD uFlags,     // Type of name specified in lpszName
 LPITEMIDLIST *ppidlOut      // Pointer to new ITEMIDLIST
);
```

Parameters

hwndOwner
Handle of the owner window that the client should specify if it displays a dialog box or message box.

pidl
Pointer to an ITEMIDLIST structure that uniquely identifies the file object or subfolder relative to the parent folder.

lpszName
Pointer to a null-terminated string that specifies the new display name.

uFlags
Value indicating the type of name specified by the *lpszName* parameter. For a list of possible values, see the description of the **SHCONTF** enumerated type.

ppidlOut
Address in which the method returns a pointer to the new **ITEMIDLIST** structure. This parameter can be NULL, and in that case, the method does not return the new **ITEMIDLIST** for the object.

If this parameter is not NULL, this method frees the specified ITEMIDLIST structure and allocates a new one using the task allocator. The calling application is responsible for freeing the new **ITEMIDLIST** structure. If an error occurs, the method returns NULL in this address.

Return Values

Returns NOERROR if successful or an OLE-defined error value otherwise.

Remarks

SetNameOf sets the display name of the specified object. If it also changes the item identifier, then it returns the new item identifier (a *pidl*), which is allocated by the task allocator. Changing the display name of a file system object or folder within renames the file or directory.

See Also

ITEMIDLIST

APPENDIX T
OCGUIDE.DOC

OLE Control and Control Container Guidelines

V2.0 Preliminary

13th December 1995

Distribution: Public

DRAFT

I.Contents

NOTE: THIS DOCUMENT IS AN EARLY RELEASE OF THE FINAL SPECIFICATION. IT IS MEANT TO SPECIFY AND ACCOMPANY SOFTWARE THAT IS STILL IN DEVELOPMENT. SOME OF THE INFORMATION IN THIS DOCUMENTATION MAY BE INACCURATE OR MAY NOT BE AN ACCURATE REPRESENTATION OF THE FUNCTIONALITY OF THE FINAL SPECIFICATION OR SOFTWARE. MICROSOFT ASSUMES NO RESPONSIBILITY FOR ANY DAMAGES THAT MIGHT OCCUR EITHER DIRECTLY OR INDIRECTLY FROM THESE INACCURACIES. MICROSOFT MAY HAVE TRADEMARKS, COPYRIGHTS, PATENTS OR PENDING PATENT APPLICATIONS, OR OTHER INTELLECTUAL PROPERTY RIGHTS COVERING SUBJECT MATTER IN THIS DOCUMENT. THE FURNISHING OF THIS DOCUMENT DOES NOT GIVE YOU A LICENSE TO THESE TRADEMARKS, COPYRIGHTS, PATENTS, OR OTHER INTELLECTUAL PROPERTY RIGHTS.

I.

Overview

The purpose of this document is to provide guidelines for implementing OLE controls and containers that will interoperate well with other controls and containers. This document defines the minimum set of interfaces, methods, and features that are required of OLE Controls and Containers to accomplish seamless and useful interoperability.

An OLE Control is essentially a simple OLE object that supports the IUnknown interface. It will usually support a lot more interfaces in order to offer functionality, but all additional interfaces may be viewed as optional and as such, a control container should not rely on any additional interfaces being supported. By not specifying additional interfaces that a control must support a control may efficiently target a particular area of functionality without having to support particular interfaces to qualify as a control. As always with OLE, whether in a control or a control container, it should never be assumed that an interface is available and standard return-checking conventions should always be followed. It is important for a control or control container to degrade gracefully and offer alternative functionality if an interface required is not available.

An OLE Control container must be able to host a minimal OLE Control as specified in this document, it will also support a number of additional interfaces as specified in the 'Containers' section of this document. There are a number of interfaces and methods that a container may optionally support, which are grouped into functional areas known as Component Categories. A container may support any combination of component categories, for example, a component category exists for 'Databinding' and a container may or may not support the databinding functionality, depending on the market needs of the container. If a control needs databinding support from a container to function, then it will enter this requirement in the registry. This allows a control container to only offer for insertion those controls that it knows it can successfully host. It is important to note that Component Categories are specified as part of OLE and are not specific to OLE Controls, the controls architecture uses Component Categories to identify areas of functionality that an OLE component may support. Component categories are not cumulative or exclusive, so a control container can support one category without necessarily supporting another.

It is important for controls that require optional features, or features specific to a certain container to be clearly packaged and marketed with those requirements. Similarly containers that offer certain features or component categories must be marketed and packaged as offering those levels of support when hosting OLE controls. It is recommended that controls target and test with as many containers as possible and degrade gracefully to offer less or alternative functionality if interfaces or methods are not available. In a situation where a control cannot perform its designated job function without the support of a component category, then that category should be entered as a requirement in the registry in order to prevent the control being inserted in an inappropriate container.

These guidelines define those interfaces and methods that a control may expect a control container to support, although as always a control should check the return values when using QueryInterface or other methods to obtain pointers to these interfaces. A container should not expect a control to support anything more than the IUnknown interface, and these guidelines identify what interfaces a control may support and what the presence of a particular interface means.

A. Why are the OLE Control and Control Container Guidelines Important?

OLE Controls have become the primary architecture for developing programmable software components for use in a variety of different containers ranging from software development tools to end-user productivity tools. In order for a control to operate well in a variety of containers, the control must be able to assume some minimum level of functionality that it can rely on in all containers.

By following these guidelines, control and container developers make their controls and containers more reliable and interoperable, and ultimately, better and more usable components for building component-based solutions.

This document provides guidelines towards good interoperability. It is expected that new interfaces and component categories will develop over time, future versions of this document reflecting these changes will be made readily available through Microsoft. It is important to note that this document does not cover detailed semantics of the OLE interfaces, this is covered by the SDK documentation.

A. What to do When an Interface You Need is Not Available

This section states some fundamental rules that apply to all OLE programming. OLE programs should use QueryInterface to acquire interface pointers, and must check the return value. OLE applications cannot safely assume that QueryInterface will succeed, this requirement applies to all OLE applications. If the requested interface is not available (i.e., QueryInterface returns E_NOINTERFACE), the control or container must degrade gracefully, even if that means that it cannot perform its designated job function.

A. What's New in V2.0?

This release of the guidelines embraces the concept of Component Categories which are a part of the OLE specification. In previous versions of this dicument component categories were loosely referred to as 'function groups' and were used to identify areas of functionality that a container may optionally support, for this version there has been a definition of how component categories work for OLE Controls and some fundamental categories are identified. The use of component categories allows the relaxing of some of the previous rules that identified interfaces as being mandatory, and allows greater flexibility for controls to efficiently target certain areas of functionality without having to provide superfluous additional support in order to qualify as a control. This edition of the guidelines also discusses what the presence or absence of an interface means and what to do in that situation.

The remainder of this document is divided into four sections. The first discusses guidelines for implementing controls, the second discusses guidelines for implementing control containers, the third discusses component categories, and the fourth discusses general guidelines, relevant to both control and control container developers.

I.

Controls

An OLE control is really just another term for "OLE Object" or more specifically, "COM Object." In other words, a control, at the very least, is some COM object that supports the *IUnknown* interface and is also self-registering. Through *IUnknown::QueryInterface* a container can manage the lifetime of the control as well as dynamically discover the full extent of a control's functionality based on the available interfaces. This allows a control to implement as little functionality as it needs to, instead of supporting a large number of interfaces that actually don't do anything. In short, this *minimal* requirement for nothing more than *IUnknown* allows any control to be as lightweight as it can.

In short, other than *IUnknown* and self-registration, there are no other *requirements* for a control. There are however conventions that should be followed about what the support of an interface means in terms of functionality provided to the container by the control. This section then describes what it means for a control to actually support an interface, as well as methods, properties, and events that a control should provide as a baseline if it has occasion to support methods, properties, and events.

A. Self Registration

OLE controls must support self-registration by implementing the *DllRegisterServer* and *DllUnregisterServer* functions. OLE controls must register all of the standard registry entries for embeddable objects and automation servers.

OLE Controls must use the component categories API to register themselves as a control and register the component categories that they require a host to support and any categories that the control implements, see the Component Categories section of this document. In addition an OLE Control may wish to register the 'control' keyword in order to allow older control containers such as VB4 to host them.

OLE Controls should also register the ToolBoxBitmap32 registry key, although this is not mandatory.

The Insertable component category should only be registered if the control is suitable for use as a compound document object. It is important to note that a compound document object must support certain interfaces beyond the *minimum* IUnknown required for an OLE Control. Although an OLE Control may qualify as a Compound Document Object, the control's documentation should clearly state what behavior to expect under these circumstances.

A. What Support for an Interface Means

Besides the *IUnknown* interface, an OLE Control—or COM Object for that matter—expresses whatever optional functionality it supports through additional interfaces. This is to say that *no other interfaces are required above IUnknown.* To that end, the following table lists the interfaces that an OLE Control might support, and what it means to support that interface. Further details about the member functions of these interfaces are given in a later section.

Interface	Comments/What it Means to Support the Interface
IOleObject	If the control requires communication with its client site for anything other than events (see *IconnectionPointContainer*), then *IOleObject* is a necessity. When implementing this interface, the control must also support the semantics of the following members: *SetHostNames, Close, EnumVerbs, Update, IsUpToDate, GetUserClassID, GetUserType, GetMiscStatus,* and the *Advise, Unadvise,* and *EnumAdvise* members that work in conjunction with a container's *IAdviseSink* implementation.[9]

[9] **A control implementing *IOleObject* must be able to handle *IAdviseSink* if the container provides one; a container may not, in which case a control ensures, of course, that it does not attempt to call a non-existent sink.**

Interface	**Comments/What it Means to Support the Interface**
IOleInPlaceObject	Expresses the control's ability to be in-place activated and possibly UI activated. This interface means that the control has a user interface of some kind that can be activated, and *IOleInPlaceActiveObject* is supported as well. Required members are *GetWindow, InPlaceActivate, UIDeactivate, SetObjectRects,* and *ReactivateAndUndo*. Support for this interface requires support for *IOleObject*.
IOleInPlaceActiveObject	An in-place capable object that supports *IOleInPlaceObject* must also provide this interface as well, though the control itself doesn't necessarily implement the interface directly.
IOleControl	Expresses the control's ability and desire to deal with (a) mnemonics (*GetControlInfo, OnMnemonic* members), (b) ambient properties (*OnAmbientPropertyChange*), and/or (c) events that the control requires the container to handle (*FreezeEvents*). Note that mnemonics are different than accelerators that are handled through *IOleInPlaceActiveObject*: mnemonics have associated UI and are active even when the control is not UI active. A control's support for mnemonics means that the control also knows how to use the container's *IOleControlSite::OnControlInfoChanged* member. Because this requires the control to know the container's site, support for mnemonics also means support for *IOleObject*. In addition, knowledge of mnemonics requires in-place support and thus *IOleInPlaceObject*. If a control uses any container-ambient properties, then it must also implement this interface to receive change notifications, as following the semantics of changes is required. Because ambient properties are only available through the container site's *IDispatch*, ambient property support means that the control supports *IOleObject* (to get the site) as well as being able to generate *IDispatch::Invoke* calls. The *FreezeEvents* method is necessary for controls that must know when a container is *not* going to handle an event—this is the only way for control to know this condition. If *FreezeEvents* is only necessary in isolation, such that other *IOleControl* members are not implemented, then *IOleControl* can stand alone without *IOleObject* or *IOleInPlaceObject*.
IDataObject	Indicates that the control can supply at least (a) graphical renderings of the control (CF_METAFILEPICT is the minimum if the control has any visuals at all) and/or (b) property sets, if the control has any properties to provide. The members *GetData, QueryGetData, EnumFormatEtc, DAdvise, DUnadvise,* and *EnumDAdvise* are required. Support for graphical formats other than CF_METAFILEPICT is optional.
IViewObject2	Indicates that the control has some interesting visuals when it is not in-place active. If implemented, a control must support the members *Draw, GetAdvise, SetAdvise,* and *GetExtent*.
Idispatch	Indicates that the control has either (a) custom methods, or (b) custom properties that are both available via late-binding through *IDispatch::Invoke*. This also requires that the control provides type information through other *IDispatch* members. A control may support multiple *IDispatch* implementations where only one is associated with IID_IDispatch—the others must have their own unique dispinterface identifiers. A control is encouraged to supply dual interfaces for custom method and property access, but this is optional if methods and properties exist.

Interface	**Comments/What it Means to Support the Interface**
IConnectionPointContainer	Indicates that a control supports at least one "outgoing" interface, such as events or property change notifications. All members of this interface must be implemented if this interface is available at all, including *EnumConnectionPoints* which requires a separate object with *IEnumConnectionPoints*. Support for *IConnectionPointContainer* means that the object also supports one or more related objects with *IConnectionPoint* that are available through *IConnectionPointContainer* members. Each "connection point" object itself must implement the full *IConnectionPoint* interface, including *EnumConnections*, which requires another enumerator object with the *IEnumConnections* interface.
IProvideClassInfo[2]	Indicates that the object can provide its own coclass type information directly through *IProvideClassInfo::GetClassInfo*. If the control supports the later variation *IProvideClassInfo2*, then it also indicates its ability to provide its primary source IID through *IProvideClassInfo2::GetGUID*. All members of this interface must be implemented.
ISpecifyPropertyPages	Indicates that the control has property pages that it can display such that a container can coordinate this control's property pages with other control's pages when property pages are to be shown for a multi-control selection. All members of this interface must be implemented when support exists.
IPerPropertyBrowsing	Indicates the control's ability to (a) provide a display string for a property, (b) provide pre-defined strings and values for its properties and/or (c) map a property dispID to a specific property page. Support for this interface means that support for properties through *IDispatch* as above is provided. If (c) is supported, then it also means that the object's property pages mapped through *IPerPropertyBrowsing::MapPropertyToPage* themselves implement *IPropertyPage2* as opposed to the basic *IPropertyPage* interface.
IPersistStream	See "Persistence Interfaces" section.
IPersistStreamInit	See "Persistence Interfaces" section.
IPersistMemory	See "Persistence Interfaces" section.
IPersistStorage	See "Persistence Interfaces" section.
IPersistMoniker	See "Persistence Interfaces" section.
IPersistPropertyBag	See "Persistence Interfaces" section.
IOleCache[2}	Indicates support for container caching of control visuals. A control generally obtains caching support itself through the OLE function *CreateDataCache*. Only controls with meaningful static content should choose to do this (although it is not required). If a control supports caching at all, it should simply aggregate the data cache and expose both *IOleCache* and *IOleCache2* interfaces from the data cache.[10]

[10] **IOleCacheControl is only important if the control has an external out-of-process data source that itself implements *IDataObject* such that the control could directly connect the data source to the cache without any intervening layers. This will be exceptionally rare.**

Interface	Comments/What it Means to Support the Interface
IExternalConnection	Indicates that the control supports external links to itself; that is, the control is not marked with OLEMISC_CANTLINKINSIDE and supports *IOleObject::SetMoniker* and *IOleObject::GetMoniker.* A container will never query for this interface itself nor call it directly as calls are generated from inside OLE's remoting layer.
IRunnableObject	Indicates that the control differentiates being "loaded" from being "running", as some in-process objects do.

A.Persistence Interfaces

Objects that have a "persistent state" of any kind must implement at least one *IPersist** interface, and preferably multiple interfaces, in order to provide the container with the most flexible choice of how it wishes to save a control's state.

If a control has *any persistent state whatsoever,* it must, as a minimum, implement either *IPersistStream* or *IPersistStreamInit* (the two are mutually exclusive and shouldn't be implemented together for the most part). The latter is used when a control wishes to know when it is created new as opposed to reloaded from an existing persistent state (*IPersistStream* does not have the "created new" capability). The existence of either interface indicates that the control can save and load its persistent state into a stream, that is, an instance of *IStream.*

Beyond these two stream-based interfaces, the *IPersist** interfaces listed in the following table can be optionally provided in order to support persistence to locations other than an expandable *IStream.*

A set of component categories is identified to cover the support for persistency interfaces see the 'Component Categories' section of this document.

Interface	Usage
IPersistMemory	The object can save and load its state into a fixed-length sequential byte array (in memory).
IPersistStorage	The object can save and load its state into an *IStorage* instance. Controls that wish to be marked "Insertable" as other compound document objects (for insertion into non-control aware containers) *must* support this interface.
IPersistPropertyBag	The object can save and load its state as individual properties written to *IPropertyBag* which the container implements. This is used for "Save As Text" functionality in some containers.
IPersistMoniker	The object can save and load its state to a location named by a moniker. The control calls *IMoniker::BindToStorage* to retrieve the storage interface it requires, such as *IStorage, IStream, ILockBytes, IDataObject,* etc.

While support for *IPersistPropertyBag* is optional, it is strongly recommended as an optimization for containers with "Save As Text" features, such as Visual Basic.

With the exception of *IPersistStream[Init]::GetSizeMax* and *IPersistMemory::GetSizeMax,* all methods of each interface must be fully implemented.

A.Optional Methods

Implementing an interface doesn't necessarily mean implementing all member functions of that interface to do anything more than return E_NOTIMPL or S_OK as appropriate. The following table identifies the methods of the interfaces listed in the 'What Support for an Interface Means' section that a control may implement in this manner. Check with the SDK OLE Reference documentation for full syntax and valid return values from these methods. Any method not listed here must be fully implemented if the interface is supported.

Method	Comments
IOleControl	

Method	Comments
GetControlInfo, OnMnemonic	Mandatory for controls with mnemonics.
OnAmbientPropertyChange	Mandatory for controls that use ambient properties.
FreezeEvents	See 'Event Freezing' in the General Guidelines section.
IOleObject	
SetMoniker	Mandatory if the control is not marked with OLEMISC_CANTLINKINSIDE
GetMoniker	Mandatory if the control is not marked with OLEMISC_CANTLINKINSIDE
InitFromData	Optional
GetClipboardData	Optional
SetExtent	Mandatory only for DVASPECT_CONTENT
GetExtent	Mandatory only for DVASPECT_CONTENT
SetColorScheme	Optional
DoVerb	See Note 1.
IoleInPlaceObject	
ContextSensitiveHelp	Optional
ReactivateAndUndo	Optional
IOleInPlaceActiveObject	
ContextSensitiveHelp	Optional
IViewObject2	
Freeze	Optional
Unfreeze	Optional
GetColorSet	Optional
IpersistStream[Init], IPersistMemory	
GetSizeMax	See Note 2.

Notes:

1. A control with property pages must support IOleObject::DoVerbs for the OLEIVERB_PROPERTIES and OLEIVERB_PRIMARY verbs. A control that can be active must support IOleObject::DoVerbs for the OLEIVERB_INPLACEACTIVATE verb. A control that can be UI active must also support IOleObject::DoVerbs for the OLEIVERB_UIACTIVATE verb.
2. If a control supports IPersistStream[Init] and can return an accurate value, then it should do so.

A.Class Factory Options

An OLE Control, by virtue of being a COM object, must have associated server code that supports control creation through *IClassFactory* as a minimum.

It is optional, not required, that this class object also supports *IClassFactory2* for licensing management. Only those vendors that are concerned about licensing need to support COM's licensing mechanism. In other words, because *IClassFactory2* is the only way to achieve COM-level licensing, this interface is required on the class object for those controls that wish to be licensed.

A.Properties

Although most controls do have properties, controls are not required to expose any properties and thus the control does not require *IDispatch*. If the control does have properties, there are no requirements for which properties a control must expose.

A.Methods (via IDispatch and Other dispinterfaces)

Although most controls do expose and support several methods, controls are not required to expose or support any methods and thus the control does not require *IDispatch*. If the control does have any methods, there are no requirements for which methods a control must expose.

A.Events

Although most controls do expose and fire several events, controls are not required to expose or fire any events and thus the control does not require *IConnectionPointContainer.* If the control does have any events, there are no requirements for which events a control must expose.

A.Property Pages

Support for property pages and per-property browsing is strongly recommended, but not required. If a control does implement property pages, then those pages should conform to one of the standard sizes: 250x62 or 250x110 dialog units (DLUs).

A.Ambient Properties

If a control supports any ambient properties at all, it must at least respect the values of the following ambient properties under the conditions stated in the following table using the standard dispids.

Ambient Property	**Dispid**	**Comment/Conditions for Use**
LocaleID	-705	If Locale is significant to the control, e.g. for text output
UserMode	-709	If the control behaves differently in user (design) mode and run mode
UIDead	-710	If the control reacts to UI events, then it should honor this ambient property
ShowGrabHandles	-711	If the control support in-place resizing of itself
ShowHatching	-712	If the control support in-place activation and UI activation
DisplayAsDefault	-713	Only if the control is marked OLEMISC_ACTSLIKEBUTTON (which means that support for keyboard mnemonics is provided, thus *IoleControl::GetControlInfo* and *IOleControl::OnMnemonic must be implemented*).

As described previously, use of ambients requires both *IOleControl* (for *OnAmbientPropertyChange* as a minimum) as well as *IOleObject* (for *SetClientSite* and *GetClientSite*).

The OLEMISC_SETCLIENTSITEFIRST bit may not necessarily be supported by a container. In these circumstances, a control must resort to default values for the ambient properties that it requires.

A.Using the Container's Functionality

The previous sections have described some of the necessary caller-side support that an OLE Control must have in order to access certain features of its container. The following table describes a control's usage of container-side interfaces and when such usage would occur.

Interface	**Container Object**	**Usage**
IOleClientSite	Site	Controls that implement *IOleObject* call *IOleClientSite* members as part of the standard OLE embedding protocol, specifically the members *SaveObject, ShowObject, OnShowWindow* (only if a separate-window activation state is supported), *RequestNewObjectLayout,* and *GetContainer* (if communication with other controls is desired). The *GetMoniker* member is only used when the control can be linked to externally, that is, the control is not marked with OLEMISC_CANTLINKINSIDE.
IOleInPlaceSite	Site	Controls that have an in-place activate and possibly a UI active state will call *IOleInPlaceSite* members (generally all of them with the exception of *ContextSensitiveHelp*) as part of the standard OLE in-place activation protocol.
IAdviseSink	Site	Control calls *OnDataChange* if the control supports *IDataObject, OnViewChange* if the control supports *IViewObject2,* and *OnClose, OnSave,* and *OnRename* if the control supports *IOleObject.*

Interface	**Container Object**	**Usage**
IOleControlSite	Site	If supported, control calls *OnControlInfoChanged* when mnemonics change, *LockInPlaceActive* and *TransformCoords* if events are fired (the latter member is only used if coordinates are passed as event arguments), *OnFocus* and *TranslateAccelerator* if the control has a UI active state, and *GetExtendedControl* if the control wants to look at extended-control (container-owned) properties.
IDispatch (ambient properties)	Site	Used to access ambient properties.
IPropertyNotifySink	Varies	A control must generate *OnChanged* and *OnRequestEdit* for any control properties that are marked as **[bindable]** and **[request]**, respectively.
Other event sink interfaces	Varies	A control that has outgoing interfaces other than *IPropertyNotifySink* will be handed other interface pointers of the correct IID to the control's *IConnectionPoint::Advise* implementations (which are usually found in sub-objects of the control). A control always knows how to call its own event interfaces since the control defines those interfaces.
IOleInPlaceFrame	Frame	Used when a control has an in-place UI active state that requires frame-level tools or menu items.
IOleInPlaceUIWindow	Document	Used only when a control has an in-place UI active state that requires document-level or pane-level UI tools. This is rare.

I.

Containers

An OLE control container is an OLE container that supports the following additional features:

1. Embedded objects from in-process servers
2. In Place activation
3. OLEMISC_ACTIVATEWHENVISIBLE
4. Event Handling

OLE Control Containers must provide support for all of these features.

The following sections describe the specific interfaces, methods, and other features that are required of OLE Control Containers. Required Interfaces, Optional Methods, Misc. Status Bits Support, Keyboard Handling, Storage Interfaces, Ambient Properties, Extended Properties, Events, Methods, Message Reflection, and Automatic Clipping. The last section describes how to gracefully degrade when a particular control interface is not supported.

A.Required Interfaces

The table below lists the OLE Control Container interfaces, and denotes which interfaces are optional, and which are mandatory and must be implemented by control containers.

Interface	Support Mandatory?	Comments
IOleClientSite	Yes	
IAdviseSink	No	Only when the container desires (a) data change notifications (controls with *IDataObject*), (b) view change notification (controls that are not active and have *IViewObject[2]*), and (c) other notifications from controls acting as standard embedded objects.
IOleInPlaceSite	Yes	
IOleControlSite	Yes	
IOleInPlaceFrame	Yes	
IOleContainer	Yes	See Note 1.
IDispatch for ambient properties	Yes	See Note 2 and "Ambient Properties" section
Control Event Sets	Yes	See Note 2.
ISimpleFrameSite	No	ISimpleFrameSite and support for nested simple frames is optional.
IPropertyNotifySink	No	Only needed for containers that (a) have their own property editing UI which would require updating whenever a control changed a property itself or (b) want to control **[requestedit]** property changes and other such data-binding features.
IErrorInfo	Yes	Mandatory if container supports dual interfaces. See Note 2.
IClassFactory2	No	Support is strongly recommended.

Notes:

1. IOleContainer is implemented on the document or form object (or appropriate analog) that holds the container sites. Controls use IOleContainer to navigate to other controls in the same document or form.
2. Support for dual interfaces is not mandatory, but is strongly recommended. Writing OLE Control Containers to take advantage of dual interfaces will afford better performance with controls that offer dual interface support.

OLE control containers must support OLE Automation exceptions. If a control container supports dual interfaces, then it must capture automation exceptions through IErrorInfo.

A.Optional Methods

An OLE component can implement an interface without implementing all the semantics of every method in the interface, instead returning E_NOTIMPL or S_OK as appropriate. The following table describes those methods that an OLE control container is not required to implement (i.e. the control container can return E_NOTIMPL).

The table below describes optional methods; note that the method must still exist, but can simply return E_NOTIMPL instead of implementing "real" semantics. Note that any method from a mandatory interface that is not listed below must be considered mandatory and may not return E_NOTIMPL.

Method	**Comments**
IOleClientSite	
SaveObject	Necessary for persistence to be successfully supported.
GetMoniker	Necessary only if the container supports linking to controls within its own form or document.
IOleInPlaceSite	
ContextSensitiveHelp	Optional
Scroll	May return S_FALSE with no action.
DiscardUndoState	Can return S_OK with no action.
DeactivateAndUndo	Deactivation is mandatory; Undo is optional.
IOleControlSite	
GetExtendedControl	Necessary for containers that support extended controls.
ShowPropertyFrame	Necessary for containers that wish to include their own property pages to handle extended control properties in addition to those provided by a control.
TranslateAccelerator	May return S_FALSE with no action.
LockInPlaceActive	Optional
IDispatch (Ambient properties)	
GetTypeInfoCount	Necessary for containers that support non-standard ambient properties.
GetTypeInfo	Necessary for containers that support non-standard ambient properties.
GetIDsOfNames	Necessary for containers that support non-standard ambient properties.
IDispatch (Event sink)	
GetTypeInfoCount	The control knows its own type information, so it has no need to call this.
GetTypeInfo	The control knows its own type information, so it has no need to call this.
GetIDsOfNames	The control knows its own type information, so it has no need to call this.
IOleInPlaceFrame	
ContextSensitiveHelp	
GetBorder	Necessary for containers with toolbar UI (which is optional)
RequestBorderSpace	Necessary for containers with toolbar UI (which is optional)
SetBorderSpace	Necessary for containers with toolbar UI (which is optional)
InsertMenus	Necessary for containers with menu UI (which is optional)
SetMenu	Necessary for containers with menu UI (which is optional)
RemoveMenus	Necessary for containers with menu UI (which is optional)
SetStatusText	Necessary only for containers that have a status line
EnableModeless	Optional
TranslateAccelerator	Optional
IOleContainer	
ParseDisplayName	Only if linking to controls or other embeddings in the container is supported, as this is necessary for moniker binding.
LockContainer	As for ParseDisplayName
EnumObjects	Returns all OLE Controls through an enumerator with IEnumUnknown, but not necessarily all objects (since there's no guarantee that all objects are OLE controls; some may be regular Windows controls).

A.Miscellaneous Status Bits Support

OLE Control Containers must recognize and support the following OLEMISC_ status bits:

Status Bit	Support Mandatory?	Comments
ACTIVATEWHENVISIBLE	Yes	
IGNOREACTIVATEWHENVISIBLE	No	Needed for inactive and windowless control support. See Note 1.
INSIDEOUT	No	Not generally used with OLE Controls but rather with compound document embeddings. Note this is contrary to some SDK documentation that says this bit must be set for the ACTIVATEWHENVISIBLE bit to be set.
INVISIBLEATRUNTIME	Yes	Designates a control that should be visible at design time, but invisible at run time.
ALWAYSRUN	Yes	
ACTSLIKEBUTTON	No	Support is normally mandatory although it is not necessary for document style containers.
ACTSLIKELABEL	No	Support is normally mandatory although it is not necessary for document style containers.
NOUIACTIVATE	Yes	
ALIGNABLE	No	
SIMPLEFRAME	No	See 'SimpleFrameSite Containment' in the Component Categories section.
SETCLIENTSITEFIRST	No	Support for this bit is recommended but not mandatory.
IMEMODE	No	

Notes:

1. The IGNOREACTIVATEWHENVISIBLE bit is for containers hosting inactive and windowless controls. The IGNOREACTIVATEWHENVISIBLE bit is introduced as part of the OLE Controls 96 specification, see this documentation for more details.

A.Keyboard Handling

Keyboard handling support for the following functionality is strongly recommended, although it is recognized that it is not applicable to all containers.

- Support for OLEMISC_ACTSLIKELABEL and OLEMISC_ACTSLIKEBUTTON status bits.
- Implementing the DisplayAsDefault ambient property (if it exists, it can return FALSE).
- Implementing tab handling, including tab order.

Some containers will use OLE controls in traditional compound document scenarios. For example, a spreadsheet may allow a user to embed an OLE control into a worksheet. In such scenarios, the container would do keyboard handling differently, because the keyboard interface should remain consistent with the user's expectations of a spreadsheet. Documentation for such products should inform users of differences in control handling in these different scenarios. Other containers should endeavor to honor the above functionality correctly, including Mnemonic handling.

A.Storage Interfaces

Control containers must be able to support controls that implement *IPersistStorage*, *IPersistStream*, or *IPersistStreamInit*. Optionally, a container can support any other persistence interfaces such as *IPersistMemory*, *IPersistPropertyBag*, and *IPersistMoniker* for those controls that provide support.

Once an OLE Control Container has chosen and initialized a storage interface to use (*IPersistStorage, IPersistStream, IPersistStreamInit*, etc), that storage interface will remain the primary storage interface for the lifetime of the control, i.e. the control will remain in possession of the storage. This does not preclude the container from saving to other storage interfaces.

OLE Control Containers do not need to support a "save as text" mechanism, thus using *IPersistPropertyBag* and the associated container-side interface *IPropertyBag* are optional.

A.Ambient Properties

At a minimum, OLE control containers must support the following ambient properties using the standard dispids.

Ambient Property	Dispid	Comments/Conditions
LocaleID	-705	
UserMode	-709	For containers that have different user and run environments.
DisplayAsDefault	-713	For those containers where a default button is relevant.

A.Extended Properties, Events and Methods

OLE Control Containers are not required to support extended controls. However, if the control container does support extended properties, then it must support the following minimal set:

Visible
Parent
Default
Cancel

Currently, extended properties, events, and methods do not have standard dispids.

A.Message Reflection

It'is strongly recommended that an OLE control container supports message reflection. This will result in more efficient operation for subclassed controls. If message reflection is supported, the MessageReflect ambient property must be supported and have a value of TRUE. If a container does ***not*** implement message reflection, then the OLE CDK creates ***two*** windows for ***every*** sub-classed control, to provide message reflection on behalf on the control container.

A.Automatic Clipping

It is strongly recommended that an OLE control container supports automatic clipping of its controls. This will result in more efficient operation for most controls. If automatic clipping is supported, the AutoClip ambient property must be supported and have a value of TRUE.

Automatic clipping is the ability of a container to ensure that a control's drawn output goes only to the container's current clipping region. In a container that supports automatic clipping, a control can paint without regard to its clipping region, because the container will automatically clip any painting that occurs outside the control's area. If a container does not support automatic clipping, then CDK-generated controls will create an extra parent window if a non-null clipping region is passed.

A.Degrading Gracefully In the Absence of an Interface

Because a control may not support any interface other than *IUnknown*, a container has to degrade gracefully when it encounters the absence of any particular interface.

One might question the usefulness of a "control" with nothing more than *IUnknown*. But consider the advantages that a control receives from a container's visual programming environment (such as VB) when the container recognizes the object as a "control":

1. A button for the object appears in a toolbox.
2. One can create an object by dragging it from the toolbox onto a form.
3. One can give the object a name that is recognized in the visual programming environment.
4. The same name in (3) above can be used immediately in writing any other code for controls on the same form (or even a different form).
5. The container can automatically provide code entry points for any events available from that object.
6. The container provides its own property browsing UI for any available properties.

When an object isn't recognized as a "control", then it potentially loses all of these very powerful and beneficial integration features. For example, in Visual Basic 4.0 it is very difficult to really integrate some random object that is not a "control" in the complete sense, but may still have properties and events. Because VB 4's idea of a control is very restrictive the object does not gain any of the integration features above. But even a control with *IUnknown*, where the mere *lifetime* of the control determines the *existence* of some resource, should be able to gain the integration capabilities described above.

As current tools require a large set of control interfaces to gain any advantage, controls are generally led to *over-implementation*, such that they contain more code than they really need. Controls that could be 7K might end up being 25K, which is a big performance problem in areas such as the Internet. This has also led to the perception that one can only implement a control with one tool like the CDK because of the complexity of implementing *all* the interfaces—and this has implications when a large DLL like OC30.DLL is required for such a control, increasing the working set. If not all interfaces are required, then this opens up many developers to writing very small and light controls with straight OLE or with other tools as well, minimizing the overhead for each control.

This is why this document recognizes a "control" as any object with a CLSID and an *IUnknown* interface. Even with nothing more than *IUnknown*, a container with a programming environment should be able to provide at least features #3 and #4 from the list above. If the object provides a ToolBoxBitmap32 registry entry, it gains #1 and #2. If the object supplies *IConnectionPointContainer* (and *IProvideClassInfo* generally) for some event set, it gains #5, and if it supports *IDispatch* for properties and methods, it gains #6, as well as better code integration in the container.

In short, an object should be able to implement as little as *IDispatch* and one event set exposed through *IConnectionPointContainer* to gain all of those visual features above.

With this in mind, the following table describes what a container might do in the absence of any possible interface. Note that only those interfaces are listed that the container will directly obtain through *QueryInterface*. Other interfaces, like *IOleInPlaceActiveObject*, are obtained through other means.

Interface	**Meaning of Interface Absence**
IViewObject2	The control has no visuals that it will draw itself, so has no definite extents to provide. In run-time, the container simply doesn't attempt to draw anything when this interface is absent. In design time, the container must at least draw some kind of default rectangle with a name in it for such a control, so a user in a visual programming environment can select the object and check out its properties, methods, and events that exist. Handling the absence of *IViewObject2* is critical for good visual programming support.
IOleObject	The control doesn't need the site whatsoever, nor does it take part in any embedded object layout negotiation. Any information (like control extents) that a container might expect from this interface should be filled in with container-provided defaults.
IOleInPlaceObject	The control doesn't go in-place active (like a label) and thus never attempts to activate in this manner. Its only activation may be its property pages.
IOleControl	Control has no mnemonics and no use of ambient properties, and doesn't care if the container ignores events. In the absence of this interface, the container just doesn't call its members.
IDataObject	The control provides no property sets nor any visual renderings that could be cached, so the container would choose to cache some default presentation in the absence of this interface (support for CF_METAFILEPICT, specifically) and disable any property-set related functionality.
IDispatch	The control has no custom properties or methods. The container does not need to try to show any control properties in this case, and should disallow any custom method calls that the container doesn't recognize as belonging to its own extended controls (that may support methods and properties). As extended controls generally delegate certain *IDispatch* calls to the control, an extended control should not expect the control to have *IDispatch* at all.
IConnectionPointContainer	The control has no events, so the container doesn't have to think about handling any.
IProvideClassInfo[2]	The control either doesn't have type information or events, or the container needs to go into the control's type information through the control's registry entries. The existence of this interface is an optimization.
ISpecifyPropertyPages	The control has no property pages, so if the container has any UI that would invoke them, the container should disable that UI.
IPerPropertyBrowsing	The control has no display name itself, no predetermined strings and values, and no property to page mapping. This interface is nearly always used for generating container user interface, so such UI elements would be disabled in the absence of this interface.
*IPersist**	The control has no persistent state to speak of, so the container doesn't have to worry

	about saving any control-specific data. The container will, of course, save its own information about the control in its own form or document, but the control itself has nothing to contribute to that information.
IOleCache[2]	The object doesn't support caching. A container can still support caching by just creating a data cache itself using *CreateDataCache*.

I.Component Categories

OLE's component categories allow a software component's abilities and requirements to be identified by entries in the registry. In a scenario where a container may not wish to or not be able to support an area of functionality, such as databinding for example, the container will not wish to host controls that require databinding in order to perform their job function. Component Categories allow areas of functionality such as databinding to be identified, so that the control container can avoid those controls that state it to be a requirement. Component Categories are specified separately as part of OLE and are not specific to the OLE Control architecture, the specification for component categories includes a set of APIs for manipulation of the component category registry keys.

A.What are Component Categories and how do they work?

Component Categories identify those areas of functionality that a software component supports and requires, a registry entry is used for each category or identified area of functionality. Each component category is identified by a globally unique identifier (GUID), when a control is installed it registers itself as a control in the system registry using the component category ID for control, see the 'Self Registration' section. Within the control's self registration it will also register those component categories that it implements and those component categories that it requires a container to support in order to successfully host the control.

When a control container is offering controls to the user to insert, it only allows the user to select and instantiate those controls that will be able to function adequately in that environment. For example, if the control container does not support databinding, then the container will not allow the user to select and instantiate those controls that have an entry in the registry signifying that they require the databinding component category. A common dialog for control insertion and APIs to handle the registry entries are available.

Component categories are not cumulative or exclusive, a control can require any mix of component categories to function. A control that has no required entries for component categories may be expected to be capable of functioning in any control container and not require any specific functionality of a control container to function.

The following component categories are identified here, where necessary more detailed specifications of the categories may be available.

- ISimpleFrameSite control containment.
- Simple Databinding through the IPropertyNotifySink interface.
- Advanced Databinding (as supported by the additional databinding interfaces of VB4.0).
- Visual Basic private interfaces - IVBFormat, IVBGetControl
- Internet aware controls.
- Windowless controls.

This is **not** a definitive list of categories; further categories are likely to be defined in the future as new requirements are identified. An up-to-date list of component categories is available from Microsoft on their world wide web site, this list reflects those component categories that have been identified by Microsoft and any others that about which vendors have informed Microsoft.

It is important to remember that controls should attempt to work in as many environments as possible. If it is possible, the control should degrade its functionality when placed in a container that does not support certain interfaces. The purpose of component categories is to prevent a situation where the control is placed in an environment that is unsuitable and the control can not achieve its desired task. Generally, a control should degrade gracefully when interfaces are not present, a control may choose to advise the user with a message box that some functionality is not available or clearly document the functionality required of a control container for optimal performance.

Note older controls and containers do not make use of Component Categories and instead rely on the 'control' keyword being present against the control in the registry. In order to be recognized by older containers controls may wish to register the 'control' keyword in the registry, control developers should check that the control can successfully be hosted in such containers before doing this. Containers that use component categories may successfully use them to host older controls as the components category DLL handles the mapping, a separate category exists for older controls CATID_ControlV1 so that a container may optionally exclude them if necessary.

As Component Categories are identified by GUIDs it is possible for containers that offer particular specific functionality to have their own category IDs, generated using a GUID generation tool. However this can possibly undermine the advantage of interoperability of controls and containers so it is preferred that wherever possible existing component categories be used. Vendors are encouraged to consult together when defining new component categories to ensure that they meet the common requirements of the marketplace, and follow the spirit of interoperability of OLE Controls.

A.SimpleFrameSite Containment

A container control is an OLE control that is capable of containing other controls. A group box that contains a collection of radio buttons is an example of a container control. Container controls should set the OLEMISC_SIMPLEFRAME status bit, and should call its container's ISimpleFrameSite implementation. An OLE control container that supports Container Controls must implement ISimpleFrameSite.
CATID - {157083E0-2368-11cf-87B9-00AA006C8166} CATID_SimpleFrameControl

A.Simple Data Binding

The OLE Controls Architecture defines a data-binding mechanism, whereby an OLE Control can specify that one or more of its properties are bindable. In most cases, a data-bound control should not absolutely require data binding, so that it could be inserted into a container that does not support data binding. Obviously, in such a situation, the functionality of the control may be reduced.
CATID - {157083E1-2368-11cf-87B9-00AA006C8166} CATID_PropertyNotifyControl

A.Advanced Data Binding

There is a set of advanced data binding interfaces that allow a more complex databinding scenario to be supported. This component category covers that area of functionality.
CATID - {157083E2-2368-11cf-87B9-00AA006C8166} CATID_VBDataBound

A.Visual Basic private interfaces

Two interfaces that are implemented by Visual Basic are identified here for component categories. It is not expected that controls should require these categories as it is possible for controls to offer alternative functionality when these are not available.
The IVBFormat interface allows controls to better integrate into the Visual Basic environment when formatting data.
CATID - {02496840-3AC4-11cf-87B9-00AA006C8166} CATID_VBFormat
The IVBGetControl interface allows a control to enumerate other controls on the VB form.
CATID - {02496841-3AC4-11cf-87B9-00AA006C8166} CATID_VBGetControl

A.Internet-Aware Objects

There are certain categories identified to cover the persistency interfaces, these have been identified as a result of defining how controls function across the internet. A container that does not support the full range of persistency interfaces should ensure that it does not host a control that requires a combination of interfaces that it does not support. Details of the features required for internet aware controls are available in the 'OLE Controls - COM objects for the internet' specification.

The following tables describe the meaning for various categories as both "implemented" and "required" categories.

"Required" Categories	**Description**
CATID_PersistsToMoniker,	Each of these categories are mutually exclusive and are only used when an object supports only one persistence mechanism at all (hence the mutual exclusion). Containers that do not support the persistence mechanism described by one of these categories should prevent themselves from creating any objects of classes so marked.
CATID_PersistsToStreamInit,	
CATID_PersisitsToStream,	
CATID_PersistsToStorage,	
CATID_PersistsToMemory,	
CATID_PersistsToFile,	
CATID_PersistsToPropertyBag	
CATID_RequiresDataPathHost	The object *requires* the ability to save data to one or more paths and

requires container involvement, therefore requiring container support for *IBindHost.*

"Implemented" Categories	**Description**
CATID_PersistsToMoniker, CATID_PersistsToStreamInit, CATID_PersistsToStream, CATID_PersistsToStorage, CATID_PersistsToMemory, CATID_PersistsToFile, CATID_PersistsToPropertyBag	Object supports the corresponding *IPersist** mechanism for the category.

The following table provides the exact CATIDs assigned to each category:

Category	**CATID**
CATID_RequiresDataPathHost	0de86a50-2baa-11cf-a229-00aa003d7352
CATID_PersistsToMoniker	0de86a51-2baa-11cf-a229-00aa003d7352
CATID_PersistsToStorage	0de86a52-2baa-11cf-a229-00aa003d7352
CATID_PersistsToStreamInit	0de86a53-2baa-11cf-a229-00aa003d7352
CATID_PersistsToStream	0de86a54-2baa-11cf-a229-00aa003d7352
CATID_PersistsToMemory	0de86a55-2baa-11cf-a229-00aa003d7352
CATID_PersistsToFile	0de86a56-2baa-11cf-a229-00aa003d7352
CATID_PersistsToPropertyBag	0de86a57-2baa-11cf-a229-00aa003d7352

A. Windowless Controls

The OLE Controls 96 specification includes a definition for 'windowless' controls. Such controls do not operate in their own window and require a container to offer a shared window into which the control may draw, see the 'OLE Controls 96' specification. Windowless controls are designed to be compatible with older control containers by creating their own window in that situation, windowless control containers should host windowed controls in the traditional way with no problem. It may however be useful for a container to distinguish those controls that can operate in a windowless mode, so an appropriate component category is defined.
CATID - {1D06B600-3AE3-11cf-87B9-00AA006C8166} CATID_WindowlessObject

I.

General Guidelines

This section describes various features, hints and tips for OLE control and OLE control container developers to help ensure good interoperability between controls and control containers.

A.Overloading IPropertyNotifySInk

Many OLE Control Containers implement a modeless property browsing window. If a control's properties are altered through the control's property pages, then the control's properties can get out of sync with the container's view of those properties (the control is always right, of course). To ensure that it always has the current values for a control's properties, an OLE Control Container can overload the IPropertyNotifySink interface (data binding) and also use it to be notified that a control property has changed. This technique is optional, and is not required of OLE Control Containers or OLE controls.

Note that a control should use IPropertyNotifySink::OnRequestEdit only for data binding; it is free to use OnChanged for either or both purposes.

A.Container-Specific Private Interfaces

Some containers provide container-specific private interfaces for additional functionality or improved performance. Controls that rely on those container-specific interfaces should, if possible, work without those container-specific interfaces present so that the control functions in different containers. For example, Visual Basic® implements private interfaces that provide string formatting functionality to controls. If a control makes use of VB's private formatting interfaces, it should be able to run with default formatting support if these interfaces are not available. If the control can function without the private interfaces, it should take appropriate action (such as warn the user of reduced functionality) but continue to work. If this is not an option, then a component category should be registered as required to ensure that only containers supporting this functionality can host these controls.

A.Multi-Threaded Issues

Starting with Microsoft® Windows® 95 and Microsoft® Windows NT™ 3.51, OLE provides support for multi-threaded applications, allowing applications to make OLE calls from multiple threads. This multi-threaded support is called the "apartment model", it is important that all OLE components using multiple threads follow this model. The apartment model requires that interface pointers are marshaled (using CoMarshallInterface, and CoUnmarshalInterface) when passed between threads. For more information about apartment model threading, refer to the Win32 SDK documentation, and the OLEAPT sample (in Win32® SDK).

A.Event Freezing

A container can notify a control that it is not ready to respond to events by calling IOleControl::FreezeEvents(TRUE). It can un-freeze the events by calling IOleControl::FreezeEvents(FALSE). When a container freezes events, it is freezing *event processing*, not *event receiving*; that is, a container can still receive events while events are frozen. If a container receives an event notification while its events are frozen, the container should ignore the event. No other action is appropriate.

A control should take note of a container's call to IOleControl::FreezeEvents(TRUE) if it is important to the control that an event is not missed. While a container's event processing is frozen, a control should implement one of the following techniques:

1. Fire the events in the full knowledge that the container will take no action.
2. Discard all events that the control would have fired.
3. Queue up all pending events and fire them after the container has called IOleControl::FreezeEvents(FALSE).
4. Queue up only relevant or important events and fire them after the container has called IOleControl::FreezeEvents(FALSE).

Each technique is acceptable and appropriate in different circumstances. The control developer is responsible for determining and implementing the appropriate technique for the control's functionality.

A.Container Controls

As described above, container controls are OLE Controls that visually contain other controls. The OLE Controls Architecture specifies the ISimpleFrameSite interface to enable container controls. Containers may also support container controls without supporting ISimpleFrameSite, although the behavior cannot be guaranteed. For this reason, a component category exists for SimpleFrameSite controls where the full functionality of this interface is required.

In order to support container controls without implementing ISimpleFrameSite, an OLE Control Container must:

- Activate all controls at all times.
- Reparent the contained controls to the hWnd of the containing control.
- Remain the parent of the container control.

A.WS_GROUP and WS_TABSTOP Flags In Controls

A control should not use the WS_GROUP and WS_TABSTOP flags internally; some containers rely on these flags to manage keyboard handling.

A.Multiple Controls In One DLL

A single .OCX DLL can container any number of OLE controls, thus simplifying the distribution and use of a set of related controls.

If you ship multiple controls in a single DLL, be sure to include the vendor name in ***each*** control name in the package. Including the vendors' names in each control name will enable users to easily identify controls within a package. For example, if you ship a DLL that implements three controls, Con1, Con2 and Con3, then the control names should be:

<Your company name> Con1 Control
<Your company name> Con2 Control
<Your company name> Con3 Control

A.IOleContainer::EnumObjects

This method is used to enumerate over all the OLE objects contained in a document or form, returning an interface pointer for each OLE object. The container must return pointers to each OLE object that shares the same container. Nested forms or nested controls must also be enumerated.

Some containers implement "extender controls", which wrap non-OLE controls, and then return pointers to these extender controls as a form is enumerated. This behavior enables OLE controls and OLE control containers to integrate well with non-OLE controls, and is thus recommended, but not required.

A.Enhanced Metafiles

Not surprisingly, enhanced metafiles provide more functionality than standard metafiles; using enhanced metafiles generally simplifies rendering code. An enhanced metafile DC is used in exactly the same way as a standard metafile DC. Enhanced metafiles are not available in 16-bit OLE. OLE supports enhanced metafiles, and includes backwards compatibility with standard metafiles and 16-bit applications.

32-bit OLE control containers should use enhanced metafiles instead of standard metafiles.

A.Licensing

In order to embed licensed controls successfully, OLE control containers must use IClassFactory2 instead of IClassFactory. Several OLE creation and loading helper functions (i.e., OleLoad and CoCreateInstance) explicitly call IClassFactory and not IClassFactory2, and therefore cannot be used to create or load licensed OLE controls. OLE Control Containers should explicitly create and load OLE controls using IClassFactory2. In the future, Microsoft will update these standard APIs to use both

IClassFactory and IClassFactory2, as appropriate.

A.Dual Interfaces

OLE Automation enables an object to expose a set of methods in two ways: via the IDispatch interface, and through direct OLE Vtable binding. IDispatch is used by most tools available today, and offers support for late binding to properties and methods. Vtable binding offers much higher performance because this method is called directly instead of through IDispatch::Invoke. IDispatch offers late bound support, where direct Vtable binding offers a significant performance gain; both techniques are valuable and important in different scenarios. By labeling an interface as "dual" in the type library, an OLE Automation interface can be used either via IDispatch, or it can be bound to directly. Containers can thus choose the most appropriate technique. Support for dual interfaces is strongly recommended for both controls and containers.

A.IPropertyBag and IPersistPropertyBag

IPropertyBag and IPersistPropertyBag optimize "save as text" mechanisms, and therefore are recommended for OLE control containers that implement a "save as text" mechanism. IPropertyBag is implemented by a container, and is roughly analogous to IStream. IPersistPropertyBag is implemented by controls, and is roughly analogous to IPersistStream.

PRINT OF DRAWINGS
AS ORIGINALLY FILED

395/774

1/7

08/761699

96503T' 66919280

FIG. 1

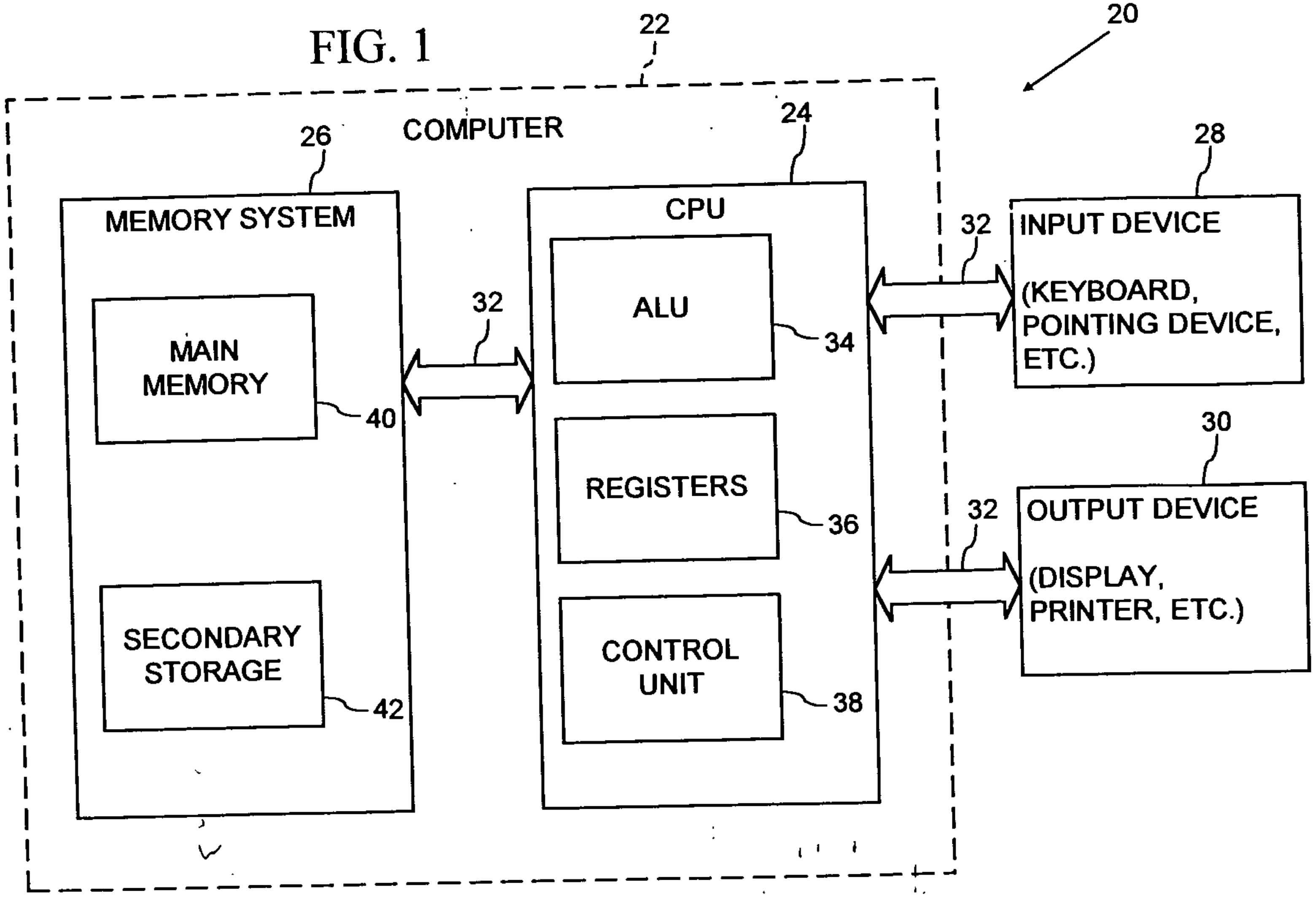
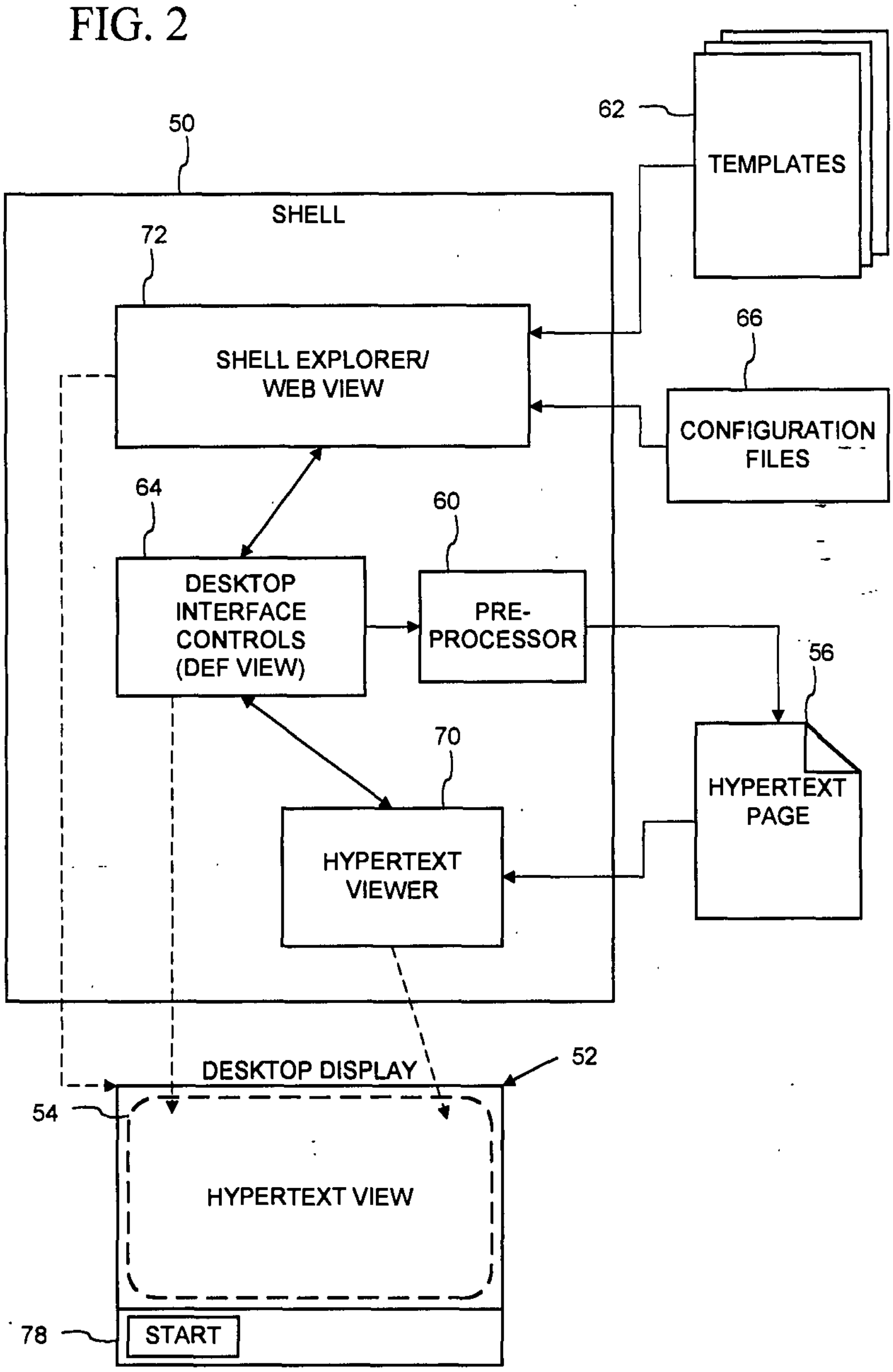


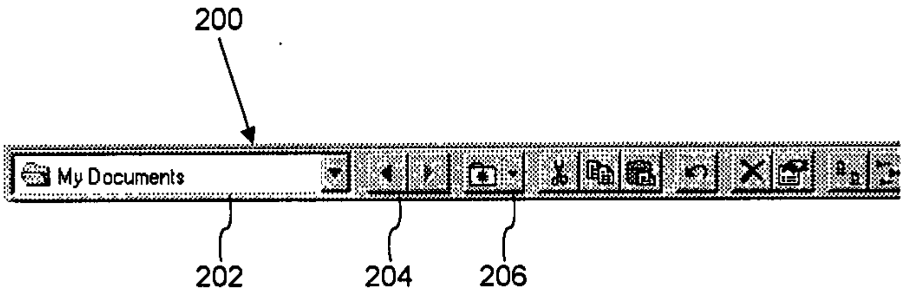
FIG. 2


50
SHELL
72
SHELL EXPLORER/ WEB VIEW
62
TEMPLATES
66
CONFIGURATION FILES
64
DESKTOP INTERFACE CONTROLS (DEF VIEW)
60
PRE-PROCESSOR
56
HYPERTEXT PAGE
70
HYPERTEXT VIEWER
DESKTOP DISPLAY
52
54
HYPERTEXT VIEW
78
START

PRINT OF DRAWINGS
AS ORIGINALLY FILED

08/761699

FIG. 3



PRINT OF DRAWINGS
AS ORIGINALLY FILED

08/761699

FIG. 4

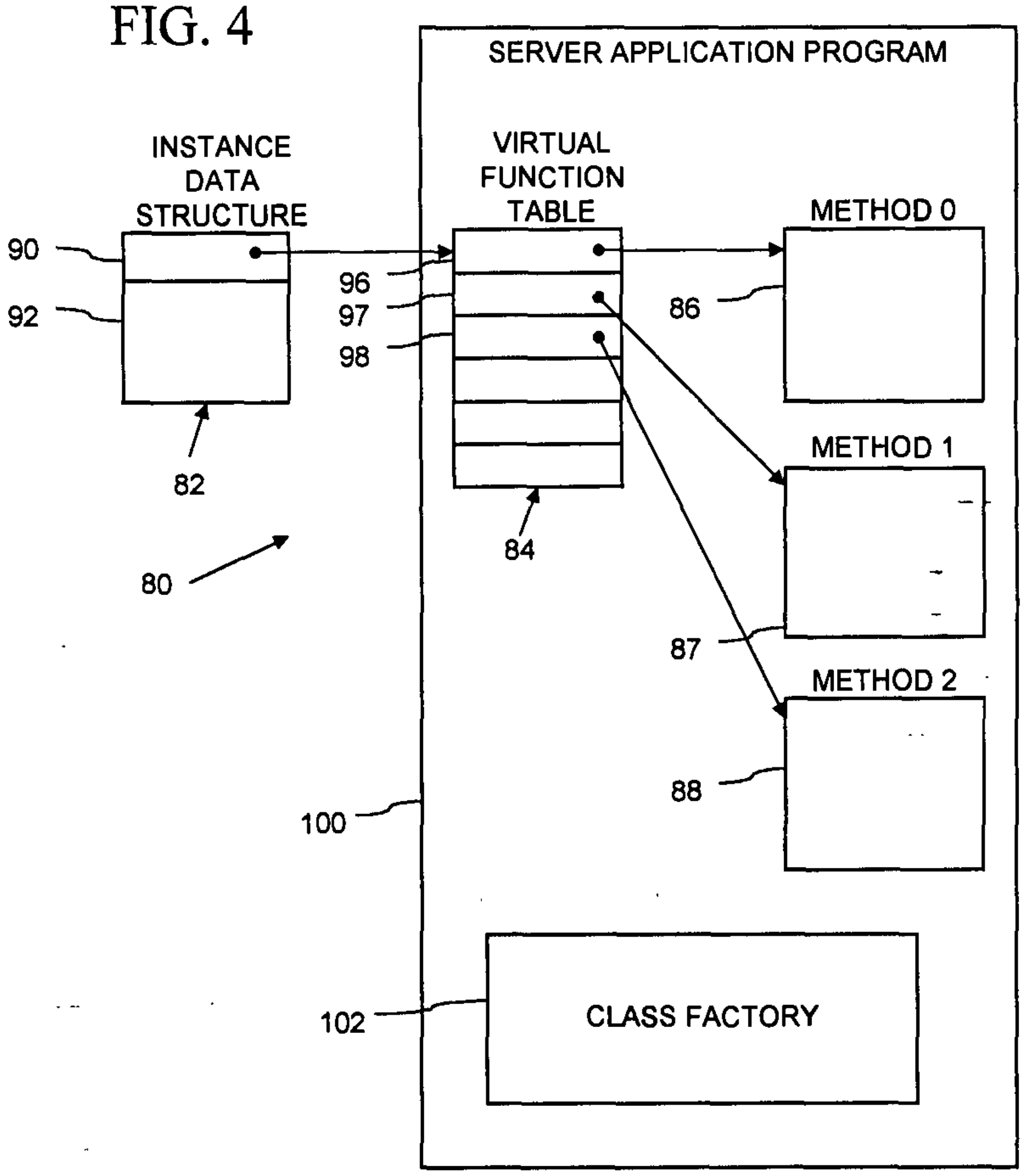
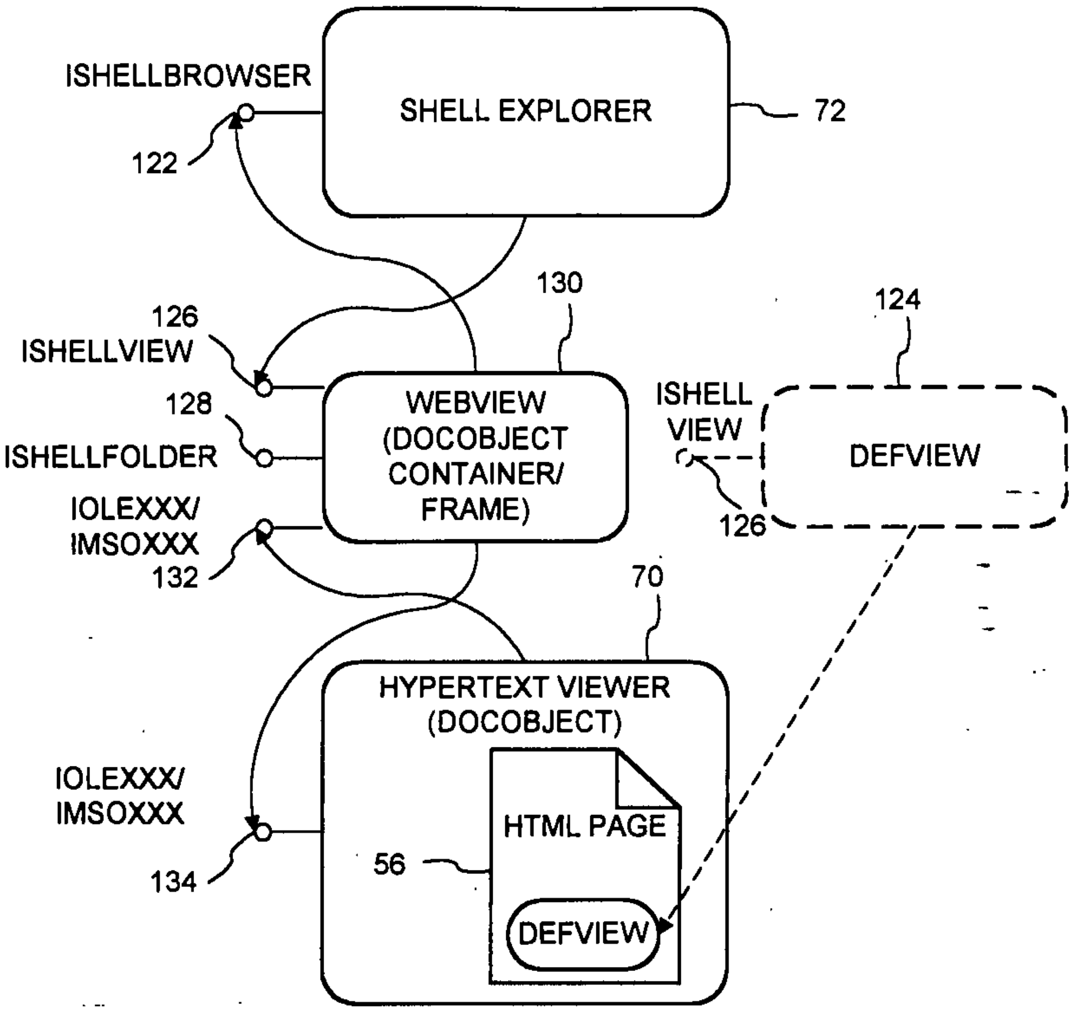


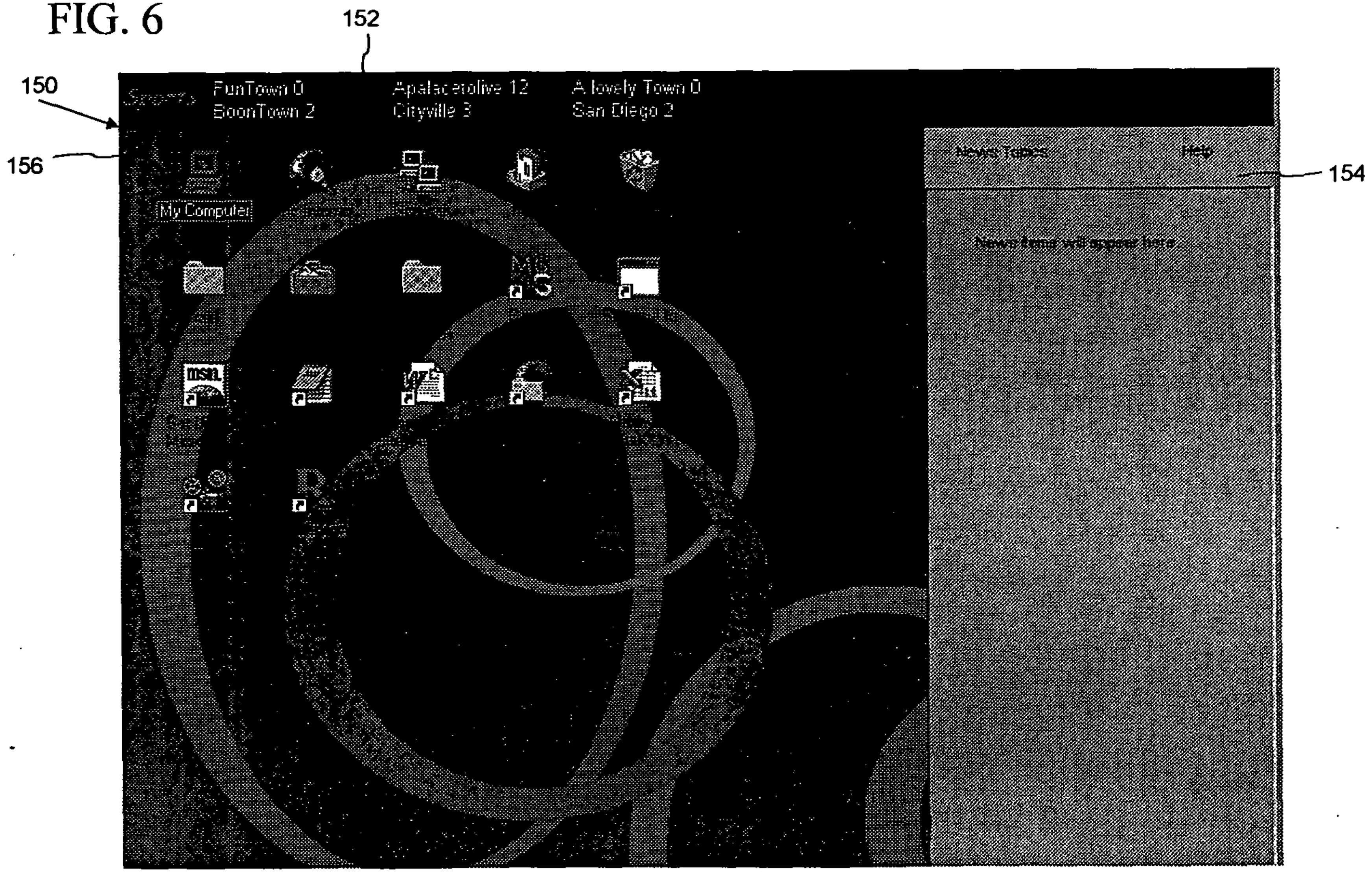
FIG. 5


ISHELLBROWSER
SHELL EXPLORER
72
122
130
124
126
ISHELLVIEW
128
ISHELLFOLDER
WEBVIEW
(DOCOBJECT
CONTAINER/
FRAME)
ISHELL
VIEW
DEFVIEW
126
IOLEXXX/
IMSOXXX
132
70
HYPERTEXT VIEWER
(DOCOBJECT)
HTML PAGE
IOLEXXX/
IMSOXXX
56
134
DEFVIEW

FRONT OF DRAWINGS
AS ORIGINALLY FILED

08/761699

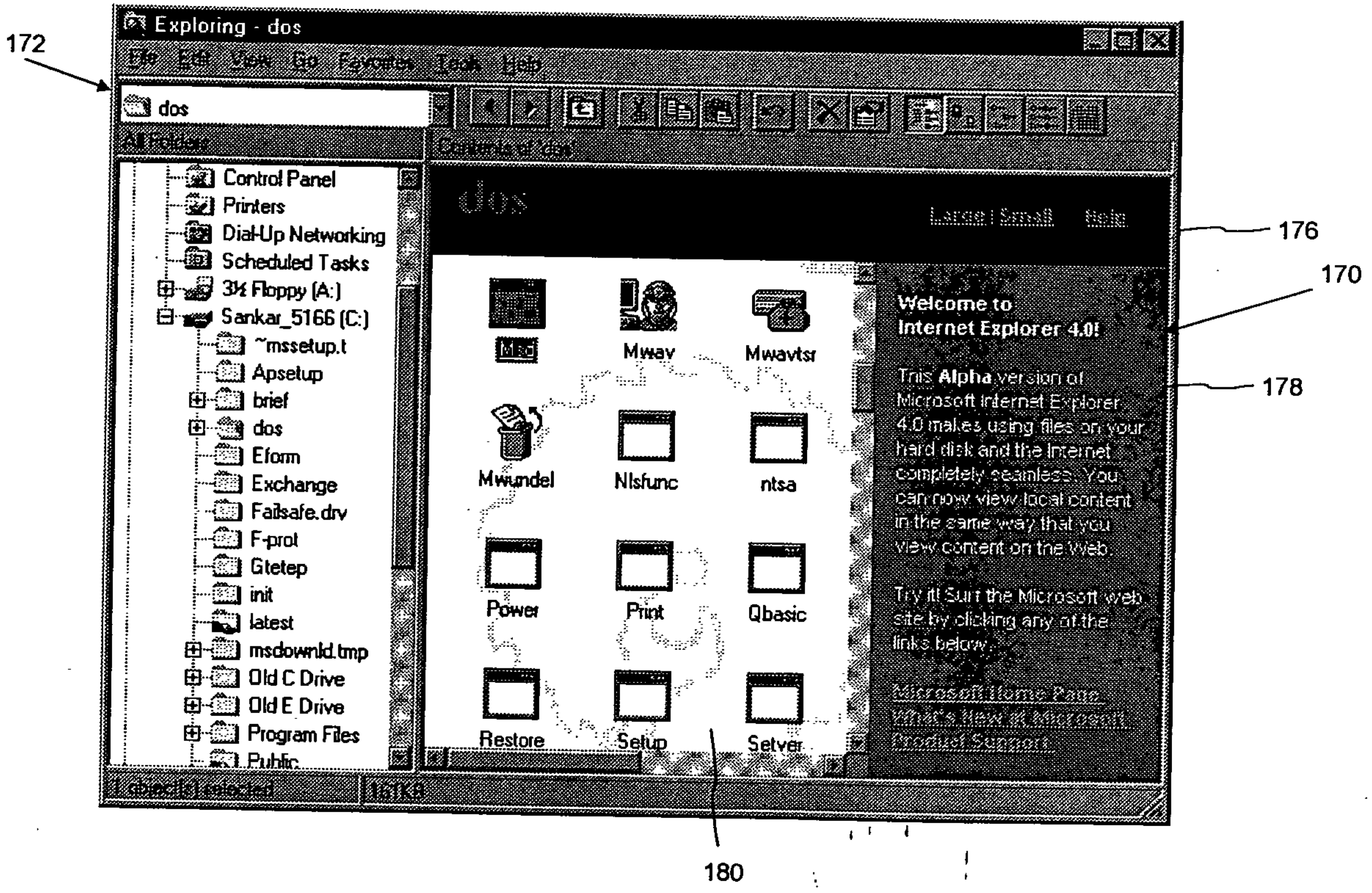
FIG. 6



PRINT OF DRAWINGS
AS ORIGINALLY FILED

08/761699

FIG. 7



Welcome to Internet Explorer 4.0!

This **Alpha** version of Microsoft Internet Explorer 4.0 makes using files on your hard disk and the Internet completely seamless. You can now view local content in the same way that you view content on the Web.

Try it! Surf the Microsoft web site by clicking any of the links below.

Microsoft Home Page
What's New at Microsoft
Product Support

SAW/jkj 12/6/96 3382- \MS82104

0 /761699 PATENT

Express Mail Label No. EM126586705US

Attorney's Ref. No. 3382-45418

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BOX PATENT APPLICATION
ASSISTANT COMMISSIONER FOR PATENTS
Washington, D.C. 20231

Transmitted herewith for filing is the patent application of:

Inventors: Benjamin W. Slivka,Teresa Martineau, Christopher Ralph Brown, George Pitt, Satoshi Nagajima, Sankar Ramasubtamanian, Mike Sheldon

For: OPERATING SYSTEM SHELL WITH HYPERTEXT DESKTOP

Enclosed are:

[X] 34 pages of specification, 4 pages of claims, an abstract, an unsigned Combined Declaration and Power of Attorney and 177 pages of appendices.

[X] 7 sheet(s) of formal drawings.

CLAIMS AS FILED						
For	Number Filed		Number Extra	Rate		Basic Fee $770.00
Total claims	17	-20	0	x $22.00	=	0.00
Independent Claims	7	-3	4	x 80.00	=	320.00
TOTAL FILING FEE						$1090.00

DATE OF DEPOSIT: December 6, 1996

[X] The Commissioner is hereby authorized to charge any additional fees which may be required in connection with the filing of this application and recording any assignment filed herewith, or credit over-payment, to Account No. 02-4550. A copy of this sheet is enclosed.

[X] A check in the amount of $1090.00 to cover [X] filing fee is enclosed.

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
cc: Ms. Shari Davidson
Mark M. Meininger, Esq.
Docketing Secretary

\MS\45418\45418PA.TRS

Respectfully submitted,

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP

By Stephen A Wight
Stephen A. Wight
Registration No. 37,759

08/761699

SAW/jkj 12/6/96 3382-45 \IS82104

PATENT

Express Mail Label No. EM126586705US

Attorney's Ref. No. 3382-45418

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Box PATENT APPLICATION
ASSISTANT COMMISSIONER FOR PATENTS
Washington, D.C. 20231

Transmitted herewith for filing is the patent application of:
Inventors: Benjamin W. Slivka,Teresa Martineau, Christopher Ralph Brown, George Pitt, Satoshi Nagajima, Sankar Ramasubtamanian, Mike Sheldon
For: OPERATING SYSTEM SHELL WITH HYPERTEXT DESKTOP
Enclosed are:

[X] 34 pages of specification, 4 pages of claims, an abstract, an unsigned Combined Declaration and Power of Attorney and 177 pages of appendices.

[X] 7 sheet(s) of formal drawings.

CLAIMS AS FILED						
For	Number Filed		Number Extra	Rate		Basic Fee $770.00
Total claims	17	-20	0	x $22.00	=	0.00
Independent Claims	7	-3	4	x 80.00	=	320.00
TOTAL FILING FEE						$1090.00

DATE OF DEPOSIT: December 6, 1996

[X] The Commissioner is hereby authorized to charge any additional fees which may be required in connection with the filing of this application and recording any assignment filed herewith, or credit over-payment, to Account No. 02-4550. A copy of this sheet is enclosed.

[X] A check in the amount of $1090.00 to cover [X] filing fee is enclosed.

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
cc: Ms. Shari Davidson
Mark M. Meininger, Esq.
Docketing Secretary

\MS\45418\45418PA.TR9

Respectfully submitted,

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP

By Stephen a Wight
Stephen A. Wight
Registration No. 37,759

TRANSMITTAL - PAGE 1

UNITED STATI)EPARTMENT OF COMMERCE
Patent and Tr..demark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

#2

APPLICATION NUMBER	FILING/RECEIPT DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NO./TITLE
[illegible]	12/06/96	[illegible]	[illegible]

[illegible]

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON
ONE WORLD TRADE CENTER SUITE 1600
121 S.W. SALMON STREET
PORTLAND OR 97204-2988

NOT ASSIGNED

0000

DATE MAILED: 02/10/97

NOTICE TO FILE MISSING PARTS OF APPLICATION
Filing Date Granted

An Application Number and Filing Date have been assigned to this application. However, the items indicated below are missing. The required items and fees identified below must be timely submitted ALONG WITH THE PAYMENT OF A SURCHARGE for items 1 and 3-6 only of $ 130 for a ☑ large entity □ small entity in compliance with 37 CFR 1.27. The surcharge is set forth in 37 CFR 1.16(e). Applicant is given TWO MONTHS FROM THE DATE OF THIS NOTICE within which to file all required items and pay any fees required above to avoid abandonment. Extensions of time may be obtained by filing a petition accompanied by the extension fee under the provisions of 37 CFR 1.136(a).

If all required items on this form are filed within the period set above, the total amount owed by applicant as a ☑ large entity □ small entity (verified statement filed), is $ 130 .

□ 1. The statutory basic filing fee is:
- □ missing.
- □ insufficient.

Applicant must submit $________ to complete the basic filing fee and/or file a verified small entity statement claiming such status (37 CFR 1.27).

□ 2. Additional claim fees of $________, including any multiple dependent claim fees, are required. *Applicant must either submit the additional claim fees or cancel additional claims for which fees are due.*

□ 3. The oath or declaration:
- □ is missing.
- □ does not cover the newly submitted items.
- □ does not identify the application to which it applies.
- □ does not include the city and state or foreign country of applicant's residence.

An oath or declaration in compliance with 37 CFR 1. 63, including residence information and identifying the application by the above Application Number and Filing Date is required.

☑ 4. The signature(s) to the oath or declaration is/are:
- ☑ missing.
- □ by a person other than inventor or person qualified under 37 CFR 1.42, 1.43, or 1.47.

A properly signed oath or declaration in compliance with 37 CFR 1.63, identifying the application by the above Application Number and Filing Date, is required.

□ 5. The signature of the following joint inventor(s) is missing from the oath or declaration:

____

An oath or declaration listing the names of all inventors and signed by the omitted inventor(s), identifying this application by the above Application Number and Filing Date, is required.

□ 6. A $________ processing fee is required since your check was returned without payment (37 CFR 1.21(m)).

□ 7. Your filing receipt was mailed in error because your check was returned without payment.

□ 8. The application does not comply with the Sequence Rules. *See attached "Notice to Comply with Sequence Rules 37 CFR 1.821-1.825."*

□ 9. OTHER:

Direct the response and any questions about this notice to "Attention: Box Missing Parts."

A copy of this notice MUST be returned with the response.

Greene
Customer Service Center
Initial Patent Examination Division (703) 308-1202

PART 3 - OFFICE COPY

S AW:jab 4/10/97 3382-45418 MS 4

PATENT

Attorney's Matter No. 3382-45418/SAW

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

Slivka et al.

Application No. 08/761,699

Filed: December 6, 1996

For: OPERATING SYSTEM SHELL WITH HYPERTEXT DESKTOP

Examiner:

Date: April 10, 1997

CERTIFICATE OF MAILING

I hereby certify that this paper and the documents referred to as being attached or enclosed herewith are being deposited with the United States Postal Service on April 10, 1997 as First Class Mail in an envelope addressed to: BOX MISSING PARTS, ASSISTANT COMMISSIONER FOR PATENTS, WASHINGTON, D.C. 20231.

Stephen A Wight
Attorney for Applicant

TRANSMITTAL LETTER

ASSISTANT COMMISSIONER FOR PATENTS
Washington, DC 20231

Enclosed for filing in the above-referenced application are the following:

(x) Copy of Notice to File Missing Parts of Application--Filing Date Granted dated 12/06/96
(x) Combined Declaration and Power of Attorney executed by applicant
(x) Surcharge for Late Filing of Fee or Oath or Declaration in the amount of $130.00
(x) A check in the amount of $130.00 to cover the above-listed fees.
(x) The Commissioner is hereby authorized to charge any additional fees or credit over-payment, to Account No. 02-4550 which may be required. A copy of this sheet is enclosed.

Respectfully submitted,

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP

By Stephen A Wight
Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446

cc: Ms. Kymerie Schmidt (LCA)
Mark M. Meininger, Esq.

#3

COMBINED DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled OPERATING SYSTEM SHELL WITH HYPERTEXT DESKTOP, the specification of which

[ ] is attached hereto.

[X] was filed on December 6, 1996 as Application No. 08/761,699 .

[ ] was described and claimed in PCT International Application No. ____________, filed on ____________, and as amended under PCT Article 19 on ____________ (if applicable).

[ ] and was amended on ________________ (if applicable).

[ ] with amendments through ____________ (if applicable).

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in Title 37, Code of Federal Regulations, § 1.56. If this is a continuation-in-part application filed under the conditions specified in 35 U.S.C. § 120 which discloses and claims subject matter in addition to that disclosed in the prior copending application, I further acknowledge the duty to disclose material information as defined in 37 CFR § 1.56 which occurred between the filing date of the prior application and the national or PCT international filing date of the continuation-in-part application.

I hereby claim foreign priority benefits under Title 35, United States Code, § 119(a)-(d) of any foreign application(s) for patent or inventor's certificate or of an PCT International application(s) designating at least one country other than the United States of America listed below and have also identified below any foreign application(s) for patent or inventor's certificate or any PCT International application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application(s) on which priority is claimed:

Prior Foreign Applications			Priority Claimed
____________	____________	____________	[ ] Yes [ ] No
(Number)	(Country)	(Day/Month/Yr. Filed)	

I hereby claim the benefit under Title 35, United States Code, § 119(e) of any United States provisional application(s) listed below:

____________	____________
(Application No.)	(Filing Date)

I hereby claim the benefit under Title 35, United States Code, § 120 of any United States application(s) or § 365(c) of any PCT International application(s) designating the United States, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT International application in the manner provided by the first paragraph of Title 35, United States Code, § 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, § 1.56(a) which occurred between the filing date of the prior application and the national or PCT International filing date of this application:

(Application No.) (Filing Date) (Status: patented, pending, abandoned)

The undersigned hereby authorizes the U.S. attorney or agent named herein to accept and follow instructions from ___________ as to any action to be taken in the Patent and Trademark Office regarding this application without direct communication between the U.S. attorney or agent and the undersigned. In the event of a change in the persons from whom instructions may be taken, the U.S. attorney or agent named herein will be so notified by the undersigned.

I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application, to file a corresponding international application, and to transact all business in the Patent and Trademark Office connected therewith:

Name	Reg. No.	Name	Reg. No.
Kenneth S. Klarquist	16,445	Donald L. Stephens Jr.	34,022
James Campbell	19,978	Stacey C. Slater	36,011
James S. Leigh	20,434	Douglas D. Hancock	35,889
Arthur L. Whinston	19,155	Garth A. Winn	33,220
David P. Petersen	28,106	Stephen A. Wight	37,759
Richard J. Polley	28,107	Joel R. Meyer	37,677
Ramon A. Klitzke II	30,188	Joseph T. Jakubek	34,190
William Y. Conwell	31,943	Mark A. Porter	35,327
Mark L. Becker	31,325	Alan E. Dow	35,123
William D. Noonan	30,878	Mark M. Meininger	32,428
John D. Vandenberg	31,312	Robert F. Scotti	39,830
Patrick W. Hughey	31,169	Gregory V. Bean	36,448
John W. Stuart	24,540	John R. Dawson	39,504

Address all telephone calls to Stephen A. Wight at telephone number (503) 226-7391.

Address all correspondence to:

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP
One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, OR 97204-2988

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of sole or first inventor: Benjamin W. Slivka

Inventor's signature ____________________ 3/7/97
Date

Residence: Clyde Hill, Washington WA

Citizenship: USA
Post Office address: 2725 96th Ave., NE, Clyde Hill, WA 98004

Anne SM.

Full name of second joint inventor, if any: Teresa Martineau

2-00 Inventor's signature ____________ 2/28/97 Date

Residence: Kirkland, Washington Wa

Citizenship: Canada

Post Office address: 13106 NE 108th St., Kirkland, WA 98033

Full name of third joint inventor, if any: Christopher Ralph Brown

Inventor's signature Chris R. Brown 4/2/97 Date

3-00 Residence: Seattle, Washington Wa

Citizenship: United States of America

Post Office address: 1340 E. Interlaken Blvd., Seattle, WA 98102

Full name of fourth joint inventor, if any: George Pitt

Inventor's signature ____________ 4/2/97 Date

227 214th Ave NE

4-00 Residence: Redmond, Washington 98053 Wa

Citizenship: U.S.A.

Post Office address: One Microsoft Way, Redmond, WA 98052

Nakajima S.N 4/8/97.

Full name of fifth joint inventor, if any: Satoshi ~~Nagajima~~

Inventor's signature ____________ 4/8/97. Date

5-00 Residence: Redmond, Washington 4902 166th Ct NE Redmond WA 98052. Wa

Citizenship:

Post Office address: One Microsoft Way, Redmond, WA 98052

Full name of sixth joint inventor, if any: Sankar Ramasubramanian

6-00 Inventor's signature Sankar Ramasubramanian 4/2/97 Date

Residence: Redmond, Washington Wa

Citizenship: India

Post Office address: 17718 NE 104th Wy., Redmond, WA 98052

Full name of seventh joint inventor, if any: Mike Sheldon

Inventor's signature ____________ 2 APR 97 Date

1316 N. Allen Pl., Seattle WA 98103

7-00 Residence: Redmond, Washington Wa

Citizenship: USA

Post Office address: One Microsoft Way, Redmond, WA 98052

130-102 0300

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

#3

APPLICATION NUMBER	FILING/RECEIPT DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NO./TITLE
[illegible]	12/06/96	SILVERA	[illegible]

[illegible]

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON
ONE WORLD TRADE CENTER SUITE 1600
121 S.W. SALMON STREET
PORTLAND OR 97204-2988

NOT ASSIGNED

0000

DATE MAILED: 02/10/97

NOTICE TO FILE MISSING PARTS OF APPLICATION
Filing Date Granted

An Application Number and Filing Date have been assigned to this application. However, the items indicated below are missing. The required items and fees identified below must be timely submitted ALONG WITH THE PAYMENT OF A SURCHARGE for items 1 and 3-6 only of $ 130 for a ☑ large entity □ small entity in compliance with 37 CFR 1.27. The surcharge is set forth in 37 CFR 1.16(e). Applicant is given TWO MONTHS FROM THE DATE OF THIS NOTICE within which to file all required items and pay any fees required above to avoid abandonment. Extensions of time may be obtained by filing a petition accompanied by the extension fee under the provisions of 37 CFR 1.136(a).

If all required items on this form are filed within the period set above, the total amount owed by applicant as a ☑ large entity □ small entity (verified statement filed), is $ 130 .

□ 1. The statutory basic filing fee is:
  □ missing.
  □ insufficient.
  Applicant must submit $ ______ to complete the basic filing fee and/or file a verified small entity statement claiming such status (37 CFR 1.27).

□ 2. Additional claim fees of $ ______, including any multiple dependent claim fees, are required.
  Applicant must either submit the additional claim fees or cancel additional claims for which fees are due.

□ 3. The oath or declaration:
  □ is missing.
  □ does not cover the newly submitted items.
  □ does not identify the application to which it applies.
  □ does not include the city and state or foreign country of applicant's residence.
  An oath or declaration in compliance with 37 CFR 1. 63, including residence information and identifying the application by the above Application Number and Filing Date is required.

☑ 4. The signature(s) to the oath or declaration is/are:
  ☑ missing.
  □ by a person other than inventor or person qualified under 37 CFR 1.42, 1.43, or 1.47.
  A properly signed oath or declaration in compliance with 37 CFR 1.63, identifying the application by the above Application Number and Filing Date, is required.

□ 5. The signature of the following joint inventor(s) is missing from the oath or declaration:
  ____
  An oath or declaration listing the names of all inventors and signed by the omitted inventor(s), identifying this application by the above Application Number and Filing Date, is required.

□ 6. A $ ______ processing fee is required since your check was returned without payment (37 CFR 1.21(m)).

□ 7. Your filing receipt was mailed in error because your check was returned without payment. 4.10.97

□ 8. The application does not comply with the Sequence Rules.
  See attached "Notice to Comply with Sequence Rules 37 CFR 1.821-1.825."

□ 9. OTHER:

Direct the response and any questions about this notice to "Attention: Box Missing Parts."

A copy of this notice MUST be returned with the response.

Coreene
Customer Service Center
Initial Patent Examination Division (703) 308-1202

PART 2 - COPY TO BE RETURNED WITH RESPONSE

SAW:jab 45418.IDS 03/06/97 MS82104

#4/Prior Art
—T. [illegible] Beth-Brown
2/26/98

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

Slivka et al.

Serial No. **08**/761,699

Filed: December 6, 1996

For: OPERATING SYSTEM SHELL WITH HYPERTEXT DESKTOP

Date: **March 6, 1997**

Art Unit

CERTIFICATE OF MAILING

I HEREBY CERTIFY THAT THIS CORRESPONDENCE IS BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE AS FIRST CLASS MAIL IN AN ENVELOPE ADDRESSED TO: ASSISTANT COMMISSIONER FOR PATENTS, WASHINGTON, D.C. 20231 ON March 6, 1997.

Stephen A Wight

Stephen A. Wight
Attorney for Applicants

INFORMATION DISCLOSURE STATEMENT (37 C.F.R. § 1.97(b)(1))

Assistant Commissioner for Patents
Washington, D.C. 20231

Sir:

This Information Disclosure Statement and the enclosed documents listed on Form PTO-1449 are being filed to comply with the Applicant's duty of disclosure under 37 C.F.R. § 1.56.

This Information Disclosure Statement is being filed within three months of the above national filing date. Thus, according to 37 C.F.R. § 1.97(b)(1), no fee is required.

It is requested that the Examiner review the enclosed information and cite this information as having been considered in connection with the present application.

Respectfully submitted,

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP

Date: March 6, 1997

By Stephen A Wight
Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204-2988
Telephone: (503) 226-7391, Ext. 641

cc: Shari Davidson (LCA)
Mark M. Meininger, Esq.

Page 1 of 1

INFORMATION DISCLOSURE STATEMENT BY APPLICANT	Docket: 3382-45418	Ser: 08/761,699
	Applicant: Slivka	
	Filed: 12/6/96	Group:

U.S. PATENT DOCUMENTS

Init.*		Number	Date	Name	Class	Sub	Filed
JLY		4,575,579	3/11/86	Simon et al.			
JLY		5,305,195	4/19/94	Murphy			
JLY		5,347,632	9/13/94	Filepp et al.			
JLY		5,491,820	2/13/96	Belove et al.			
JLY		5,572,643	11/5/96	Judson			

FOREIGN PATENT DOCUMENTS

		Number	Date	Country	Class	Sub		
JLY		EP0749081A1	24.05.96	EUROPEAN PATENT	G06F 17/60			
JLY		WO 96/30864	3.10.96	PCT	G06K 13/00			

OTHER DOCUMENTS

EXAMINER: John L. Young DATE: 3-17-99

*Examiner: Initial if considered, whether or not in conformance with MPEP 60; draw line through cite if not in conformance and not considered. Send copy.

SAW/jab 3382-45418.ids MS82104

Loc 03C0

PATENT
ATTY. REF. NO. 3382-45418

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of

Slivka et al.

Serial No.: 08/761,699

Filed: December 6, 1996

For: OPERATING SYSTEM SHELL WITH HYPERTEXT DESKTOP

Examiner:

Art Unit

CERTIFICATE OF MAILING

I HEREBY CERTIFY THAT THIS PAPER AND THE DOCUMENTS REFERRED TO AS BEING ATTACHED OR ENCLOSED HEREWITH ARE BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE ON March 6, 1997, AS FIRST-CLASS MAIL IN AN ENVELOPE ADDRESSED TO: ASSISTANT COMMISSIONER FOR PATENTS, WASHINGTON, DC 20231.

Stephen A. Wight
Stephen A. Wight
Attorney for Applicant(s)

TRANSMITTAL LETTER

TO THE ASSISTANT COMMISSIONER
FOR PATENTS:

Enclosed for filing in the above-referenced application are an Information Disclosure Statement Pursuant to 37 C.F.R. § 1.97, Form PTO-1449, and copies of the documents cited therein.

No fee is required since this document is being submitted within three months of the above national filing date.

Respectfully submitted,

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON

Date: March 6, 1997

By Stephen A. Wight
Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 SW Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391

cc: Shari Davidson (LCA)
Mark M. Meininger, Esq.

SAW/lab 3382-45418 ids MS8210

PATENT
ATTY. REF. NO. 3382-45418

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of

Slivka et al.

Serial No.: 08/761,699

Filed: December 6, 1996

For: OPERATING SYSTEM SHELL WITH HYPERTEXT DESKTOP

Examiner:

Art Unit

CERTIFICATE OF MAILING

I HEREBY CERTIFY THAT THIS PAPER AND THE DOCUMENTS REFERRED TO AS BEING ATTACHED OR ENCLOSED HEREWITH ARE BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE ON March 6, 1997, AS FIRST-CLASS MAIL IN AN ENVELOPE ADDRESSED TO: ASSISTANT COMMISSIONER FOR PATENTS, WASHINGTON, DC 20231.

Stephen A. Wight

Stephen A. Wight
Attorney for Applicant(s)

TRANSMITTAL LETTER

TO THE ASSISTANT COMMISSIONER
FOR PATENTS:

Enclosed for filing in the above-referenced application are an Information Disclosure Statement Pursuant to 37 C.F.R. § 1.97, Form PTO-1449, and copies of the documents cited therein.

No fee is required since this document is being submitted within three months of the above national filing date.

Respectfully submitted,

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON

Date: March 6, 1997

By Stephen A. Wight
Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 SW Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391

cc: Shari Davidson (LCA)
Mark M. Meininger, Esq.

GAU ~~2415~~

~~2773~~

2772

PATENT

SAW:lb 9/4/98 3382-45418; MS#82104

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

Slivka et al.

Application No. 08/761,699

Filed: December 6, 1996

For: OPERATING SYSTEM SHELL
WITH HYPERTEXT DESKTOP

Examiner:

Date: September 4, 1998

Art Unit 2415

CERTIFICATE OF MAILING

I HEREBY CERTIFY THAT THIS CORRESPONDENCE IS BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE AS FIRST CLASS MAIL IN AN ENVELOPE ADDRESSED TO: COMMISSIONER OF PATENTS AND TRADEMARKS, WASHINGTON, D.C. 20231 ON September 4, 1998.

Stephen A. Wight
Attorney for Applicant

STATUS LETTER

COMMISSIONER OF PATENTS
AND TRADEMARKS
Washington, DC 20231

A Notice of Recordation of Assignment Document was received for the above-referenced patent application on June 10, 1997. After this date, we have received nothing further with respect to the examination of this application.

Thus, a status report on the above-referenced application is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP

By Stephen A. Wight
Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391

18status.mrg Rev. 5/08/93

saw:lb 9/4/98 3382-45[illegible]; MS#82104 PATENT

cc: Microsoft Patent Group Docketing Department

KLARQUIST SPARKMAN CAMPBELL LEIGH & WHINSTON, LLP
Attorneys at Law
One World Trade Center
121 S.W. Salmon Street, Suite 1600
Portland, Oregon 97204-2988
U.S.A.

ADDRESS SERVICE REQUESTED

COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, DC 20231

08/761,699

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

ml

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR		ATTORNEY DOCKET NO.
08/761,699	12/06/96	SLIVKA	B	3382-45418

LM02/0323

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON
ONE WORLD TRADE CENTER SUITE 1600
121 SW SALMON STREET
PORTLAND OR 97204-2988

EXAMINER	
YOUNG, J	
ART UNIT	**PAPER NUMBER**
2776	5

DATE MAILED: 03/23/99

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

Commissioner of Patents and Trademarks

PTO-90C (Rev. 2/95)
*U S GPO. 1996-404-496/40510

1- File Copy

Office Action Summary	Application No. **08/761,699**	Applicant(s) **Slivka et al.**
	Examiner **John L. Young**	Group Art Unit **2776**

☒ Responsive to communication(s) filed on *Dec 6, 1996* .

☐ This action is **FINAL**.

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

A shortened statutory period for response to this action is set to expire *3* month(s), or thirty days, whichever is longer, from the mailing date of this communication. Failure to respond within the period for response will cause the application to become abandoned. (35 U.S.C. § 133). Extensions of time may be obtained under the provisions of 37 CFR 1.136(a).

Disposition of Claims

☒ Claim(s) *1-17* is/are pending in the application.

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

☐ Claim(s) ______ is/are allowed.

☒ Claim(s) *1-17* is/are rejected.

☐ Claim(s) ______ is/are objected to.

☐ Claims ______ are subject to restriction or election requirement.

Application Papers

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

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

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

☒ The specification is objected to by the Examiner.

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

Priority under 35 U.S.C. § 119

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

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

☐ received.

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

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

*Certified copies not received: ______ .

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

Attachment(s)

☒ Notice of References Cited, PTO-892

☒ Information Disclosure Statement(s), PTO-1449, Paper No(s). *4*

☐ Interview Summary, PTO-413

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

☐ Notice of Informal Patent Application, PTO-152

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

TITLE OBJECTION

1. The title of the invention is not descriptive. A new title is required that is clearly indicative of the invention to which the claims are directed. The following title is suggested: **OPERATING SYSTEM SHELL HAVING A WINDOWING GRAPHICAL USER INTERFACE WITH HYPERTEXT DESKTOP**.

This change in title "may result in [a] slightly longer [title], but the loss in brevity of title will be more than offset by the gain in its informative value in indexing, classifying, searching, etc." (See MPEP 606.01).

CLAIM REJECTIONS — 35 U.S.C. §112 ¶ 1

The following is a quotation of the first paragraph of 35 U.S.C. 112 which forms the basis of the "enablement" rejections set forth in this Office action:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his/her invention.

2. Claim 16 is unpatentable under 35 U.S.C. § 112 ¶ 1, based on lack of support in the specification and drawings. The use of "AND/OR" as alternative language in the claim without concomitant support in the specification renders the claim unpatentable.

> Claim 16, recites: "said object providing graphical icon-oriented and/or menu driven user interface elements for controlling operating system and/or file system services when the hypertext page is viewed."
>
> The examiner cannot find where the specification shows three embodiments (each separately and one together) as suggested above by the "AND/OR" language. Please show where said embodiments are found in the specification?

CLAIM REJECTIONS — 35 U.S.C. §102

The following is a quotation of 35 U.S.C. §102(a) which forms the basis of the novelty rejections set forth in this Office action:

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

3. Claims 1-4 are rejected under 35 U.S.C. §102(a) as being anticipated by Jacquelyn Gavron and Joseph Moran, How to Use Microsoft® Windows® NT 4 WORKSTATION (Emeryville: Ziff Davis Press, 1996) (herein referred to as "Gavron").

As per claim 1, Gavron shows "[a] computer having a display device, a system for displaying a user interface to an operating system as a hypertext multimedia document on the display device. . . ." (See Gavron pp. 88, 89, 108, 109, 156, 157 & 167).

Gavron shows "a graphical user interface for providing a windowing environment supporting a plurality of windows displayed on the display device according to a front to back order wherein a window toward the front in the order overlaps any windows farther back in the order which are displayed in a same area of the display device. . . ." (See Gavron pp. 24, 88 & 105).

Gavron shows "a desktop in the windowing environment of the graphical user interface for providing a full-screen view of a hypertext multimedia document displayed back-most in the order." (See Gavron pp. 102-105, 108 & 109). As disclosed above, Gavron expresses the inventive concept of the claimed invention; therefore, the elements of the instant claim read on the disclosure of Gavron cited above.

As per claim 2, Gavron shows the system of claim 1 (see the rejection of claim 1 supra).

Gavron shows "a plurality of icons in the hypertext multimedia document and operative to launch application programs on user activation." (See Gavron pp. 100-105 & 109). As disclosed above, Gavron expresses the inventive concept of the claimed invention; therefore, the elements of the instant claim read on the disclosure of Gavron cited above.

As per claim 3, Gavron shows the system of claim 1 (see the rejection of claim 1 supra).

Gavron shows "a plurality of icons in the hypertext multimedia document and operative to activate file system services." (See Gavron pp. 100-105 & 109). As disclosed above, Gavron expresses the inventive concept of the claimed invention; therefore, the elements of the instant claim read on the disclosure of Gavron cited above.

As per claim 4, Gavron shows the system of claim 1 (see the rejection of claim 1 supra).

Gavron shows "a representation in the hypertext multimedia document of the contents of a folder in a file system of the computer." (See Gavron pp. 24, 25, 28, 29, 88, 89, 104, 105, 108, 109). As disclosed above, Gavron expresses the inventive concept of the claimed invention; therefore, the elements of the instant claim read on the disclosure of Gavron cited above.

CLAIM REJECTIONS — 35 U.S.C. §103(a)

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

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

4. Claims 5, 8, 9 & 12 are rejected under 35 U.S.C. §103(a) as being unpatentable over Ed Tittel and Steve James, HTML FOR DUMMIES 2d ed., (Foster City: IDG Books Worldwide, Inc. 1996) (herein referred to as "Tittel") in view of Nakajima (5,831,606) [f/d 12/13/94].

As per claim 5, Tittel pp. 392, 397 shows "[an] operating system shell. . . ."

Tittel pp. 396, 131-133 shows a "hypertext . . . document template. . . ."

Tittel pp. 31-37 shows "hypertext multimedia . . ." extensions.

Tittel p. 388 shows a graphical "user interface. . . ." *"GUI."*

Tittel does not explicitly show "a processor for processing the template into a hypertext multimedia document having an embedded control object for providing an operating system user interface; and a viewer for displaying a view of the hypertext document."

Nakajima col. 1, ll. 11-19; and col. 1, ll. 20-26 shows *"operating systems user interfaces. . . . "*

Nakajima col. 3, ll. 31-40; cols. 87-88; and FIG. 18 shows *"a . . . viewer for viewing objects . . . on the video display. "*

Nakajima col. 2, ll. 31-44; and col. 6, ll. 11-33 shows *"a processing means for running the operating system and the shell extension handler. . . . a . . . processor application to support these features. "*

Nakajima col. 13, ll. 17-36; and col. 14, ll. 18-35 shows *"OLE embedding . . . . the shell of the operating system* [and the] *user interface. . . . "*

Nakajima proposes "viewing, processing and embedding" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Nakajima to Tittel, because addition of such modifications would have extended *"the capabilities provided by a shell of an operating system to allow an application developer to customize. . . . to better suit the needs of their users."* (See Nakajima col. 4, ll. 41-56).

As per claim 8, Tittel pp. 396, 131-133 shows a "template for use in a system to synthesize a . . . document. . . ."

Tittel pp. 31-37 shows "hypertext multimedia document. . . ." extensions.

Tittel pp. 392, 397 shows "[an] operating system. . . ."

Tittel p. 388 shows a "user interface. . . ." *"GUI."*

Tittel pp. 31-37 shows "hypertext data for incorporating multi-media information into the hypertext multimedia document. . . ."

Tittel pp. 101, 60 & 76-81 shows *"<INPUT> Input object"* tags.

Tittel does not explicitly show "a processor, an embeddable user interface control object, and a hypertext multimedia document viewer. . . ."

Nakajima col. 2, ll. 31-44; and col. 6, ll. 11-33 shows *"a processing means for running the operating system and the shell extension handler. . . . a . . . processor application to support these features."*

Nakajima col. 13, ll. 17-36; col. 14, ll. 18-35; col. 1, ll. 11-19; and col. 1, ll. 20-26 shows *"OLE embedding . . . . the shell of the operating system* [and the] *user interface. . . ."*

Nakajima col. 3, ll. 31-40; cols. 87-88; and FIG. 18 shows *"a . . . viewer for viewing objects . . . on the video display."*

Nakajima col. 2, ll. 31-44; col. 6, ll. 11-33; and FIG. 11 shows "at least one preprocessor *"means for running the operating system and the shell extension handler. . . . a . . . processor application to support these features. "*

Nakajima col. 2, ll. 31-44; col. 6, ll. 11-33; col. 3, ll. 31-40; cols. 87-88; and FIG. 18 shows "whereby the preprocessor processes the template into a hypertext multimedia document which provides an operating system user interface when displayed by the viewer."

Nakajima proposes "viewing, processing and OLE embedding" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Nakajima to Tittel, because addition of such modifications would have extended *"the capabilities provided by a shell of an operating* system *to allow an application developer to customize. . . . to better suit the needs of their users.*" (See Nakajima col. 4, ll. 41-56).

As per claim 9, Tittel pp. 14, 15, 31-37, 101, 158, 388, 392 & 397 shows a Graphical "user interface. . . . having a video screen. . . . providing an operating system user interface control. . . . providing a hypertext page having hypertext data for incorporating multi-media enhancements, and an object insertion tag indicative of said control for embedding said control. . . ." *"GUI."*

Tittel does not explicitly show "an embeddable user interface control object, and a hypertext multimedia document viewer. . . ."

Nakajima col. 13, ll. 17-36; col. 14, ll. 18-35; col. 1, ll. 11-19; and col. 1, ll. 20-26 shows *"OLE embedding . . . . the shell of the operating system* [and the] *user interface. . . . "*

Nakajima col. 3, ll. 31-40; cols. 87-88; and FIG. 18 shows *"a . . . viewer for viewing objects . . . on the video display. "*

Nakajima proposes "viewing and OLE embedding" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Nakajima to Tittel, because addition of such modifications would have extended *"the capabilities provided by a shell of an operating syst*em *to allow an application developer to customize. . . . to better suit the needs of their users.*" (See Nakajima col. 4, ll. 41-56).

As per claim 12, Tittel in view of Nakajima shows the method of claim 9 (see the rejection of claim 9 supra).

Tittel pp. 396, 131-133 shows providing a "template for use in a system to synthesize a page. . . ."

Tittel pp. 31-37 shows "hypertext data. . . . specifying a parameter. . . ."

Tittel pp. 101, 60 & 76-81 shows *"<INPUT> Input object"* tags.

Tittel does not explicitly show "a pre-processor. . . ."

Nakajima col. 2, ll. 31-44; and col. 6, ll. 11-33 shows *"a processing means for running the operating system and the shell extension handler. . . . a . . . processor application to support these features."*

Nakajima col. 2, ll. 31-44; col. 6, ll. 11-33; and FIG. 11 shows "at least one preprocessor *"means for running the operating system and the shell extension handler. . . . a . . . processor application to support these features."*

Nakajima col. 2, ll. 31-44; col. 6, ll. 11-33; col. 3, ll. 31-40; cols. 87-88; and FIG. 18 shows the "processing the template to synthesize the hypertext page. . . . and converting the parameter into hypertext data."

Nakajima proposes *"Input tag"* and "processing" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Nakajima to Tittel, because addition of such modifications would have extended *"the capabilities provided by a shell of an operating system to allow an application developer to customize. . . . to better suit the needs of their users."* (See Nakajima col. 4, ll. 41-56).

5. Claims 6, 7, 10, 11 & 13-17 are rejected under 35 U.S.C. §103(a) as being unpatentable over Tittel in view of Gavron and further in view of Nakajima (5831,606) [f/d 12/13/94].

As per claim 6, Tittel in view of Nakajima shows the operating system shell of claim 5 (see the rejection of claim 5 supra).

Tittel does not explicitly show "a view container object for providing a desktop display in which the view is embedded."

Gavron shows "a desktop display. . . ." (See Gavron pp. 3 & 13).

Nakajima col. 13, ll. 17-36; and col. 14, ll. 18-35 shows *"OLE embedding* . . . . [and the] *user interface. . . ."*

Nakajima col. 5, ll. 16-35; and col. 6, line 31 shows *"a moniker is a composite name for an object that includes a pointer to the object.* . . . [and] *icon handlers. . . ."*

Gavron proposes "desktop" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Gavron to Tittel, because addition of such modifications would have provided an *"approach* . . . [to] *setting up your desktop to look and feel the way you like."* (See Gavron p. xiii *"Introduction"*).

Nakajima proposes "icon and embedding" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring that "a view container object" would have been selected in accordance with a screen "icon." Furthermore, it would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring that to add the modifications taught by Nakajima to Tittel, because addition of such modifications would have extended *"the capabilities provided by a shell of an operating system to allow an application developer to customize. . . . to better suit the needs of their users."* (See Nakajima col. 4, ll. 41-56).

As per claim 7, Tittel in view of Nakajima shows the operating system shell of claim 5 (see the rejection of claim 5 supra).

Tittel does not explicitly show "a view container object for providing a desktop display in which the view is embedded."

Gavron shows "a desktop display. . . . application window." (See Gavron pp. 102 & 103).

Nakajima col. 13, ll. 17-36; and col. 14, ll. 18-35 shows *"OLE embedding . . . .* [and the] *user interface. . . . "*

Nakajima col. 5, ll. 16-35; and col. 6, line 31 shows *"a moniker is a composite name for an object that includes a pointer to the object. . . .* [and] *icon handlers. . . . "*

Gavron proposes "desktop application window" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Gavron to Tittel, because addition of such modifications would have provided an *"approach . . .* [to] *setting up your desktop to look and feel the way you like."* (See Gavron p. xiii *"Introduction"*).

Nakajima proposes "icon and embedding" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring that a screen *"icon"* would have been selected in accordance with "a view container object." Furthermore, it would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring that to add the modifications taught by Nakajima to Tittel, because addition of such modifications would have extended *"the capabilities provided by a shell of an operating syst*em *to allow an application developer to customize. . . . to better suit the needs of their users.*" (See Nakajima col. 4, ll. 41-56).

As per claim 10, Tittel in view of Nakajima shows the method of claim 9 (see the rejection of claim 9 supra).

Tittel pp. 350 & 351 teaches "a windowing environment on a video screen."

Tittel does not explicitly show "displaying the view in a desktop. . . ."

Gavron shows "a desktop display. . . ." (See Gavron pp. 3, 13 & 102 - 107).

Gavron proposes "desktop" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Gavron to Tittel, because addition of such modifications would have provided an *"approach* . . . [to] *setting up your desktop to look and feel the way you like."* (See Gavron p. xiii *"Introduction"*).

As per claim 11, Tittel in view of Nakajima shows the method of claim 9 (see the rejection of claim 9 supra).

Tittel pp. 350 & 351 teaches "a windowing environment on a video screen."

Tittel does not explicitly show "displaying the view in a window. . . ."

Gavron shows "displaying the view in a window. . . ." (See Gavron pp. 102 - 107).

Gavron proposes "window view" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Gavron to Tittel, because addition of such modifications would have provided an *"approach* . . . [to] *setting up your desktop to look and feel the way you like."* (See Gavron p. xiii *"Introduction"*).

As per claim 13, Tittel in view of Nakajima shows the method of claim 12 (see the rejection of claim 12 supra).

Tittel pp. 396, 131-133 shows providing a "template to use for synthesizing the hypertext page. . . ."

Tittel does not explicitly show "storing a configuration file in the folder specifying the template. . . ."

Tittel pp. 57 & 58 implicitly shows *"a descriptive markup language describes the structure and behavior of a document."* The examiner interprets this disclosure as equivalent to showing "storing a configuration file."

Tittel does not explicitly show "the view of the hypertext page represents a folder of a file system. . . ."

Gavron shows "the view of the hypertext page represents a folder of a file system. . . ." (See Gavron pp. 24, 25, 28, 29, 88, 89, 104, 105, 108 & 109).

Tittel does not explicitly show "a user navigates to the folder identifying the template. . . ."

Gavron shows "a user navigates to the folder identifying the template. . . ." (See Gavron pp. 24, 25, 28, 29, 88, 89, 104, 105, 108 & 109).

Gavron proposes *"folder"* and navigation modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Gavron to Tittel, because addition of such modifications would have provided an *"approach* . . . [to] *setting up your desktop to look and feel the way you like."* (See Gavron p. xiii *"Introduction"*).

As per claim 14, Tittel pp. 396, 131-133 shows "hypertext . . . document template[s]. . . ."

Tittel pp. 101, 60 & 76-81 shows *"<INPUT> Input object"* tags. Tittel p. 388 shows a graphical "user interface. . . ." *"GUI."*

Tittel does not explicitly show a "file system navigation method. . . ."

Gavron shows "the view of the hypertext page represents a folder of a file system. . . ." (See Gavron pp. 24, 25, 28, 29, 88, 89, 104, 105, 108 & 109).

Tittel does not explicitly show "a user navigates to the folder identifying the template. . . ."

Gavron shows "a user navigates to the folder identifying the template. . . ." (See Gavron pp. 24, 25, 28, 29, 88, 89, 104, 105, 108 & 109).

Gavron proposes *"folder"* and navigation modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Gavron to Tittel, because addition of such modifications would have provided an *"approach* . . . [to] *setting up your desktop to look and feel the way you like."* (See Gavron p. xiii *"Introduction"*).

Tittel does not explicitly show "processing the template associated with the opened folder to produce a hypertext page having the object insertion tag. . . ."

Tittel does not explicitly show "displaying a view of the hypertext page with the folder user interface control embedded therein, whereby the user manipulates the folder user interface control to activate file system services relating to the folder."

Nakajima col. 1, ll. 11-19; and col. 1, ll. 20-26 shows *"operating systems user interfaces. . . ."*

Nakajima col. 3, ll. 31-40; cols. 87-88; and FIG. 18 shows *"a . . . viewer for viewing objects . . . on the video display."*

Nakajima col. 2, ll. 31-44; and col. 6, ll. 11-33 shows *"a processing means for running the operating system and the shell extension handler. . . . a . . . processor application to support these features."*

Nakajima col. 13, ll. 17-36; and col. 14, ll. 18-35 shows *"OLE embedding. . . . the shell of the operating system* [and the] *user interface. . . ."*

Nakajima proposes "processing, embedding and displaying" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Nakajima to Tittel, because addition of such modifications would have extended *"the capabilities provided by a shell of an operating* *system to allow an application developer to customize. . . . to better suit the needs of their users*." (See Nakajima col. 4, ll. 41-56).

Gavron shows "a plurality of graphical icon-oriented [folders] and menu driven user interface elements provided by a control object, the user interface elements being manipulable by a user of the computer to initiate operating system services; and a view of a hypertext multimedia document. . . ." (See Gavron pp. 104-109 & 154-157).

Gavron proposes folder "icon" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Gavron to Tittel, because addition of such modifications would have provided an *"approach . . .* [to] *setting up your desktop to look and feel the way you like."* (See Gavron p. xiii *"Introduction"*).

As per claim 15, Tittel in view of Gavron and Nakajima shows the method of claim 14 (see the rejection of claim 14 supra).

Tittel pp. 396, 131-133 shows a "template. . . ."

Tittel p. 385 shows an example of a "default" situation.

Tittel does not explicitly show "storing configuration files in at least some of the folders each configuration containing data identifying the template. . . ."

Tittel pp. 57 & 58 implicitly shows *"a descriptive markup language describes the structure and behavior of a document."* The examiner interprets this disclosure as equivalent to showing "storing configuration files."

Tittel does not explicitly show "if the opened folder stores one of the configuration files, determining the template associated with the opened folder from the configuration file stored in the opened folder. . . ."

Tittel does not explicitly show "determining a default template is associated with the opened folder."

Gavron shows "a user navigates to the folder identifying the template. . . ." (See Gavron pp. 24, 25, 28, 29, 88, 89, 104, 105, 108 & 109).

Gavron proposes *"folder"* and navigation modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Gavron to Tittel, because addition of such modifications would have provided an *"approach . . .* [to] *setting up your desktop to look and feel the way you like."* (See Gavron p. xiii *"Introduction"*).

Nakajima col. 2, ll. 31-44; and col. 6, ll. 11-33 shows *"a processing means for running the operating system and the shell extension handler. . . . a . . . processor application to support these features."*

Nakajima proposes "template processing" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Nakajima to Tittel, because addition of such modifications would have extended *"the capabilities provided by a shell of an operating syste*m *to allow an application developer to customize. . . . to better suit the needs of their users.*" (See Nakajima col. 4, ll. 41-56).

As per claim 16, Tittel pp. 14, 15, 31-37, 101, 158, 388, 392 & 397 shows "a hypertext page having hypertext data; and an object insertion tag. . . ."

Tittel does not explicitly show "graphical icon-oriented user interface elements for controlling operating system services. . . ."

Tittel does not explicitly show "menu driven user interface elements for controlling operating system services. . . ."

Tittel does not explicitly show "graphical icon-oriented and menu driven user interface elements for controlling operating system services. . . ."

Tittel does not explicitly show "graphical icon-oriented user interface elements for controlling file system services. . . ."

Tittel does not explicitly show "graphical icon-oriented user interface elements for controlling operating system and file system services. . . ."

Tittel does not explicitly show "menu driven user interface elements for controlling file system services. . . ."

Tittel does not explicitly show "menu driven user interface elements for controlling operating system services and file system services. . . ."

Gavron shows "a plurality of graphical icon-oriented and menu driven user interface elements provided by a control object, the user interface elements being manipulable by a user of the computer to initiate operating system services; and a view of a hypertext multimedia document. . . ." (See Gavron pp. 104-109 & 154-157).

Gavron proposes "desktop in the windowing," "icon" and "menu" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Gavron to Tittel, because addition of such modifications would have provided an *"approach*. . . [to] *setting up your desktop to look and feel the way you like."* (See Gavron p. xiii *"Introduction"*).

Tittel does not explicitly show "embedding a user interface control object. . . ."

Nakajima col. 13, ll. 17-36; and col. 14, ll. 18-35 shows *"OLE embedding*. . . . [and the] *user interface. . . ."*

Nakajima col. 5, ll. 16-35; and col. 6, line 31 shows *"a moniker is a composite name for an object that includes a pointer to the object*. . . . [and] *icon handlers. . . ."*

Nakajima proposes "object and embedding" modifications that would have applied to the hypertext page system of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring that a screen *"icon"* would have been selected in accordance with "a view container object." Furthermore, it would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring that to add the modifications taught by Nakajima to Tittel, because addition of such modifications would have extended *"the capabilities provided by a shell of an operating system to allow an application developer to customize. . . . to better suit the needs of their users."* (See Nakajima col. 4, ll. 41-56).

As per claim 17, Tittel pp. 392, 397 shows "[an] operating system shell. . . ."

Tittel pp. 14, 15, 31-37, 101, 158, 388, 392 & 397 shows a Graphical "user interface. . . . having a video screen. . . . providing an operating system user interface control. . . . providing a hypertext page having hypertext data for incorporating multi-media enhancements. . . ."

Tittel does not explicitly show "a desktop in the windowing environment of the graphical user interface. . . ."

Gavron shows "a desktop in the windowing environment of the graphical user interface. . . ." (See Gavron pp. 102-105, 108, 109 & 167).

Tittel does not explicitly show "a plurality of graphical icon-oriented and menu driven user interface elements provided by a control object, the user interface elements being manipulable by a user of the computer to initiate operating system services; and a view of a hypertext multimedia document. . . ."

Gavron shows "a plurality of graphical icon-oriented and menu driven user interface elements provided by a control object, the user interface elements being manipulable by a user of the computer to initiate operating system services; and a view of a hypertext multimedia document. . . ." (See Gavron pp. 104-109 & 154-157).

Gavron proposes "icon" and "windowing" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Gavron to Tittel, because addition of such modifications would have provided an *"approach* . . . [to] *setting up your desktop to look and feel the way you like."* (See Gavron p. xiii *"Introduction"*).

Tittel does not explicitly show "control object embedding . . . for integrating the multi-media enhancements with the user interface elements."

Nakajima col. 13, ll. 17-36; and col. 14, ll. 18-35 shows *"OLE embedding* . . . . *the shell of the operating system* [and the] *user interface. . . ."*

Nakajima proposes "embedding" modifications that would have applied to the operating system shell of Tittel. It would have been obvious at the time the invention was made to one of ordinary skill in the art of Web page authoring to add the modifications taught by Nakajima to Tittel, because addition of such modifications would have extended *"the capabilities provided by a shell of an operating syst*em *to allow an application developer to customize. . . . to better suit the needs of their users*." (See Nakajima col. 4, ll. 41-56).

CONCLUSION

6. Any response to this action should be mailed to:

Commissioner of Patents and Trademarks
Washington, D.C. 20231

Any response to this action may be sent via facsimile to either:

(703) 308-9051 (for formal communications marked EXPEDITED PROCEDURE), or

(703) 308-5403 (for informal communications marked PROPOSED or DRAFT).

Hand delivered responses may be brought to:

Sixth floor Receptionist
Crystal Park II
2121 Crystal Drive
Arlington, Virginia.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to John L. Young who may be reached via telephone at (703) 305-3801. The examiner can normally be reached Monday through Friday between 8:30 A.M. and 5:00 P.M.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Michael Razavi, may be reached at (703) 305-4713.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the Group receptionist whose telephone number is (703) 305-3900.

John L. Young
Patent Examiner

March 17, 1999

Notice of References Cited

Application No. 08/761,699	Applicant Slivka et al.		
Examiner John L. Young		Group Art Unit 2776	Page 1 of 1

U.S. PATENT DOCUMENTS

		DOCUMENT NO.	DATE	NAME	CLASS	SUBCLASS
	A	5,831,606	11/3/98	Nakajima et al.,	345	326
	B					
	C					
	D					
	E					
	F					
	G					
	H					
	I					
	J					
	K					
	L					
	M					

FOREIGN PATENT DOCUMENTS

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

NON-PATENT DOCUMENTS

		DOCUMENT (Including Author, Title, Source, and Pertinent Pages)	DATE
	U	Gavron, Jacquelyn and Joseph Moran. "How to Use Microsoft (R) Windows (R) NT 4 WORKSTATION." (Emeryville: Ziff Davis Press, 1996) pp. 24, 25, 28, 29, 88, 89, 100-105, 108, 109, 154-157 & 167.	1/1/96
	V	Tittle, Ed. and Steve James. "HTML FOR DUMMIES" 2d ed. (Foster City: IDG Books Worldwide, 1996) pp. 14, 15, 31-37, 57, 58, 60, 76-81, 131-133, 158, 350, 351, 385, 388, 392, 396 & 397.	3/11/96
	W		
	X		

U. S. Patent and Trademark Office
PTO-892 (Rev. 9-95)

Notice of References Cited

Part of Paper No. 5

Form PTO 948 (Rev. 10-94) U.S. DEPARTMENT OF COMMERCE - Patent and Trademark Office Application No. 8/761699

NOTICE OF DRAFTSPERSON'S PATENT DRAWING REVIEW

PTO Draftpersons review all originally filed drawings regardless of whether they are designated as formal or informal. Additionally, patent Examiners will review the drawings for compliance with the regulations. Direct telephone inquiries concerning this review to the Drawing Review Branch, 703-305-8404.

The drawings filed (insert date) 12/6/96 are
A.____ not objected to by the Draftsperson under 37 CFR 1.84 or 1.152.
B. ✓ objected to by the Draftsperson under 37 CFR 1.84 or 1.152 as indicated below. The Examiner will require submission of new, corrected drawings when necessary. Corrected drawings must be submitted according to the instructions on the back of this Notice.

1. DRAWINGS. 37 CFR 1.84(a): Acceptable categories of drawings: Black ink Color.
   - ___ Not black solid lines. Fig(s)________
   - ___ Color drawings are not acceptable until petition is granted. Fig(s)________
2. PHOTOGRAPHS. 37 CFR 1.84(b)
   - ___ Photographs are not acceptable until petition is granted. Fig(s)________
   - ___ Photographs not properly mounted (must use brystol board or photographic double-weight paper). Fig(s)________
   - ___ Poor quality (half-tone). Fig(s)________
3. GRAPHIC FORMS. 37 CFR 1.84 (d)
   - ___ Chemical or mathematical formula not labeled as separate figure. Fig(s)________
   - ___ Group of waveforms not presented as a single figure, using common vertical axis with time extending along horizontal axis. Fig(s)________
   - ___ Individuals waveform not identified with a separate letter designation adjacent to the vertical axis. Fig(s)________
4. TYPE OF PAPER. 37 CFR 1.84(c)
   - ___ Paper not flexible, strong, white, smooth, nonshiny, and durable. Sheet(s)________
   - ___ Erasures, alterations, overwritings, interlineations, cracks, creases, and folds copy machine marks not accepted. Fig(s)________
   - ___ Mylar, velum paper is not acceptable (too thin). Fig(s)________
5. SIZE OF PAPER. 37 CFR 1.84(f): Acceptable sizes:
   21.6 cm. by 35.6 cm. (8 1/2 by 14 inches)
   21.6 cm. by 33.1 cm. (8 1/2 by 13 inches)
   21.6 cm. by 27.9 cm. (8 1/2 by 11 inches)
   21.0 cm. by 29.7 cm. (DIN size A4)
   - ___ All drawing sheets not the same size. Sheet(s)________
   - ___ Drawing sheet not an acceptable size. Sheet(s)________
6. MARGINS. 37 CFR 1.84(g): Acceptable margins:

Paper size

	21 6 cm. X 35 6 cm. (8 1/2 X 14 inches)	21.6 cm X 33.1 cm. (8 1/2 X 13 inches)	21.6 cm X 27.9 cm. (8 1/2 X 11 inches)	21.0 cm. X 29.7 cm. (DIN Size A4)
T	5 1 cm. (2")	2 5 cm. (1")	2.5 cm. (1")	2.5cm
L	64 cm. (1/4")	.64 cm. (1/4")	.64 cm. (1/4")	2.5 cm.
R	.64 cm. (1/4")	.64 cm. (1/4")	.64 cm. (1/4")	1 5 cm.
B	.64 cm. (1/4")	.64 cm. (1/4")	.64 cm (1/4")	1.0 cm.

Margins do not conform to chart above.
Sheet(s)________
____Top (T) ____ Left (L) ____Right (R) ____Bottom (B)

7. VIEWS. 37 CFR 1.84(h)
   REMINDER: Specification may require revision to correspond to drawing changes.
   - ___ All views not grouped together. Fig(s)________
   - ___ Views connected by projection lines or lead lines. Fig(s)________

   Partial views. 37 CFR 1.84(h) 2
   - ___ View and enlarged view not labled separatly or properly. Fig(s)________

   Sectional views. 37 CFR 1.84 (h) 3
   - ___ Hatching not indicated for sectional portions of an object. Fig(s)________
   - ___ Cross section not drawn same as view with parts in cross section with regularly spaced parallel oblique strokes. Fig(s)________
8. ARRANGEMENT OF VIEWS. 37 CFR 1.84(i)
   - ___ Words do not appear on a horizontal, left-to-right fashion when page is either upright or turned so that the top becomes the right side, except for graphs. Fig(s)________
9. SCALE. 37 CFR 1.84(k)
   - ___ Scale not large enough to show mechanism with crowding when drawing is reduced in size to two-thirds in reproduction. Fig(s)________
   - ___ Indication such as "actual size" or scale 1/2" not permitted. Fig(s)________
10. CHARACTER OF LINES, NUMBERS, & LETTERS. 37 CFR 1.84(l)
    - ___ Lines, numbers & letters not uniformly thick and well defined, clean, durable, and black (except for color drawings). Fig(s)________
11. SHADING. 37 CFR 1.84(m)
    - ✓ Solid black shading areas not permitted. Fig(s) 3, 6, 7
    - ___ Shade lines, pale, rough and blurred. Fig(s)________
12. NUMBERS, LETTERS, & REFERENCE CHARACTERS. 37 CFR 1.84(p)
    - ✓ Numbers and reference characters not plain and legible. 37 CFR 1.84(p)(l) Fig(s) 3, 6, 7
    - ___ Numbers and reference characters not oriented in same direction as the view. 37 CFR 1.84(p)(l) Fig(s)________
    - ___ English alphabet not used. 37 CFR 1.84(p)(2) Fig(s)________
    - ___ Numbers, letters, and reference characters do not measure at least .32 cm. (1/8 inch) in height. 37 CFR(p)(3) Fig(s)________
13. LEAD LINES. 37 CFR 1.84(q)
    - ___ Lead lines cross each other. Fig(s)________
    - ___ Lead lines missing. Fig(s)________
14. NUMBERING OF SHEETS OF DRAWINGS. 37 CFR 1.84(t)
    - ___ Sheets not numbered consecutively, and in Arabic numerals, beginning with number 1. Sheet(s)________
15. NUMBER OF VIEWS. 37 CFR 1.84(u)
    - ___ Views not numbered consecutively, and in Arabic numerals, beginning with number 1. Fig(s)________
    - ___ View numbers not preceded by the abbreviation Fig. Fig(s)________
16. CORRECTIONS. 37 CFR 1.84(w)
    - ___ Corrections not made from prior PTO-948. Fig(s)________
17. DESIGN DRAWING. 37 CFR 1.152
    - ___ Surface shading shown not appropriate. Fig(s)________
    - ___ Solid black shading not used for color contrast. Fig(s)________

COMMENTS: Gray areas Dg (Fig 3, 6, 7)

ATTACHMENT TO PAPER NO. 5 REVIEWER WS DATE 5/28/97

PTO Copy

#6
SC
8/2/99 GP 2776

SAW:mlt 7/21/99 3382-45418 MS82104

PATENT
Attorney's Matter No. 3382-45418

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:
Slivka et al.
Application No.: 08/761,699
Filed: December 6, 1996
For: OPERATING SYSTEM SHELL HAVING A HYPERTEXT DOCUMENT-BASED GRAPHICAL USER INTERFACE (AS AMENDED)
Examiner: J. Young
Date: July 23, 1999

Art Unit: 2776

CERTIFICATE OF MAILING

I hereby certify that this paper and the documents referred to as being attached or enclosed herewith are being deposited with the United States Postal Service on July 23, 1999 as First Class Mail in an envelope addressed to: ASSISTANT COMMISSIONER FOR PATENTS, WASHINGTON, D.C. 20231

Attorney for Applicant

TRANSMITTAL LETTER

TO THE ASSISTANT COMMISSIONER FOR PATENTS
Washington, D.C. 20231

Enclosed is an Amendment for the above application. The fee has been calculated as shown below.

CLAIMS AS AMENDED

For	No. after amendment	No. paid for previously		Present Extra	Rate	Fee
Total Claims	17	- 17*	=	0	$18.00	$ 0.00
Indep. Claims	7	7**	=	0	$78.00	$ 0.00
Mult. Dep. Claims Fee (if not previously paid)					$260.00	$0.00
One-month Extension of Time					$110.00	$110.00
Two-month Extension of Time					$380.00	$0.00
Three-month Extension of Time					$870.00	$0.00
TOTAL ADDITIONAL FEE FOR THIS AMENDMENT						$110.00

* greater of twenty or number for which fee has been paid.
** greater of three or number for which fee has been paid.

☒ Applicant petitions for an extension of time for the number of months indicated above. If an additional extension of time is required please consider this a petition therefor.

18/1999 SSALEEKU 00000130 08761699
C:115 110.00 OP

SAW:mlt 7/21/99 3382-45418 MS82104

PATENT
Attorney's Matter No. 3382-45418

☒ A check in the amount of $110.00 is attached.

☒ Please charge any additional fees which may be required in connection with filing this amendment and any extension of time, or credit any overpayment, to Deposit Account No. 02-4550. A copy of this sheet is enclosed.

[X] Please return the enclosed postcard to confirm that the items listed above have been received.

Respectfully submitted,

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP

By Stephen A. Wight
Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446
cc: Patent Group Docketing Dept.
Megan Thomas

SAW:mlt 7/21/99 3382-45418 MS82104

PATENT
Attorney's Matter No. 3382-45418

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

Slivka et al.

Application No.: 08/761,699

Filed: December 6, 1996

For: OPERATING SYSTEM SHELL HAVING A HYPERTEXT DOCUMENT-BASED GRAPHICAL USER INTERFACE (AS AMENDED)

Examiner: J. Young

Date: July 23, 1999

Art Unit: 2776

CERTIFICATE OF MAILING

I hereby certify that this paper and the documents referred to as being attached or enclosed herewith are being deposited with the United States Postal Service on July 23, 1999 as First Class Mail in an envelope addressed to: ASSISTANT COMMISSIONER FOR PATENTS, WASHINGTON, D.C. 20231.

Attorney for Applicant

TRANSMITTAL LETTER

TO THE ASSISTANT COMMISSIONER FOR PATENTS
Washington, D.C. 20231

Enclosed is an Amendment for the above application. The fee has been calculated as shown below.

CLAIMS AS AMENDED

For	No. after amendment	No. paid for previously		Present Extra	Rate	Fee
Total Claims	17	- 17*	=	0	$18.00	$ 0.00
Indep. Claims	7	7**	=	0	$78.00	$ 0.00
Mult. Dep. Claims Fee (if not previously paid)					$260.00	$0.00
One-month Extension of Time					$110.00	$110.00
Two-month Extension of Time					$380.00	$0.00
Three-month Extension of Time					$870.00	$0.00
TOTAL ADDITIONAL FEE FOR THIS AMENDMENT						$110.00

* greater of twenty or number for which fee has been paid.
** greater of three or number for which fee has been paid.

☒ Applicant petitions for an extension of time for the number of months indicated above. If an additional extension of time is required please consider this a petition therefor.

PATENT
Attorney's Matter No. 3382-45418

☒ A check in the amount of $110.00 is attached.

☒ Please charge any additional fees which may be required in connection with filing this amendment and any extension of time, or credit any overpayment, to Deposit Account No. 02-4550. A copy of this sheet is enclosed.

[X] Please return the enclosed postcard to confirm that the items listed above have been received.

Respectfully submitted,

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP

By Stephen A. Wight
Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446
cc: Patent Group Docketing Dept.
Megan Thomas

7/A SC 8/2/99

SAW:mlt 3382-45418 MS82104

PATENT
Attorney's Matter No. 3382-45418

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:
Slivka et al.

Art Unit 2776

Application No.: 08/761,699

Filed: December 6, 1996

For: OPERATING SYSTEM SHELL HAVING A HYPERTEXT DOCUMENT-BASED GRAPHICAL USER INTERFACE (AS AMENDED)

Examiner: J. Young

Date: July 23, 1999

CERTIFICATE OF MAILING

I hereby certify that this paper and the documents referred to as being attached or enclosed herewith are being deposited with the United States Postal Service on July 23, 1999 as First Class Mail in an envelope addressed to: ASSISTANT COMMISSIONER FOR PATENTS, WASHINGTON D.C. 20231.

Stephen A. Wight

Attorney for Applicant

Amendment

ASSISTANT COMMISSIONER FOR PATENTS
WASHINGTON, DC 20231

Responsive to the Office action dated March 23, 1999, Paper No. 5, please amend the subject application as follows:

In The Specification:

Change the title to: --OPERATING SYSTEM SHELL HAVING A HYPERTEXT DOCUMENT-BASED GRAPHICAL USER INTERFACE--.

At page 5, line 11, change "and" to --and/or--.

After "element" at page 5, line 11, add --, which control operating system and/or file system services--.

At page 19, line 6, change "controls 60" to --controls 64--.

At page 23, line 15, change "document 60" to --document 56--.

At page 27, lines 22, 26 and 30, change "controls 60" to --controls 64--.

At page 28, lines 23, 32, 33 and 34, change "controls 60" to --controls 64--

.

At page 29, lines 4, and 6, change "controls 60" to --controls 64--.

At page 31, line 23, change "control 60" to --control 64--.

At page 33, line 2, change "control 60" to --control 64--.

In the Claims:

A1

1. (Amended) In a computer having a display device, a system for displaying a user interface to an operating system as a hypertext multimedia document on the display device, comprising:

a graphical user interface for providing a windowing environment supporting a plurality of windows displayed on the display device according to a front to back order wherein a window towards the front in the order overlaps any windows farther back in the order which are displayed in a same area of the display device; and

a desktop in the windowing environment of the graphical user interface for providing a [full-screen] view of a hypertext multimedia document [displayed back-most in the order] <u>as a full-screen graphical background display over which the windows are displayed</u>.

2. (Unchanged) The system of claim 1 comprising:

a plurality of icons in the hypertext multimedia document and operative to launch application programs on user activation.

3. (Unchanged) The system of claim 1 comprising:

a plurality of icons in the hypertext multimedia document and operative to activate file system services.

4. (Unchanged) The system of claim 1 comprising:

a representation in the hypertext multimedia document of the contents of a folder in a file system of the computer.

5. (Amended) An operating system shell comprising:

A2

a hypertext multimedia document template;

a preprocessor for processing the template into a hypertext multimedia document having an embedded control object for providing at least part of an operating [system] system's graphical user interface; and

a viewer for displaying a view of the hypertext document in the graphical user interface wherein the embedded control object is manipulable by a user to activate a service of the operating system.

6. (Amended) The operating system shell of claim 5 comprising:

a view container object for hosting the viewer and providing a desktop display in which the view is embedded as a full-screen graphical background display of a windowing environment.

7. (Amended) The operating system shell of claim 5 comprising:

a view container object for hosting the viewer and providing an application window in which the view is embedded.

8. (Amended) A template for use in a system to synthesize a hypertext multimedia document to provide an operating system user interface, the system having a preprocessor, an embeddable user interface control object, and a hypertext multimedia document viewer, the template comprising:

hypertext data for incorporating multi-media information into the hypertext multimedia document;

at least one preprocessor directive for converting parameters into hypertext data; and

an object insertion tag for embedding the user interface control object into the hypertext multimedia document, wherein the embedded control object is manipulable by a user to activate a service of the operating system, whereby the preprocessor processes the template into a hypertext multimedia document which provides an operating system user interface when displayed by the viewer.

A2 Concl.

9. (Amended) A method of providing a graphical user interface to an operating system of a computer having a video screen, comprising:

providing an operating system user interface control manipulable by a user to activate a service of the operating system;

providing a hypertext page having hypertext data for incorporating multi-media enhancements, and an object insertion tag indicative of said control for embedding said control;

generating a view of the hypertext page incorporating said multi-media enhancements and with said control embedded therein; and

displaying the view on the video screen.

10. (Amended) The method of claim 9 comprising:

displaying the view in a desktop display of a windowing environment on the video screen, the desktop display being a full-screen graphical background display of the windowing environment.

11. (Unchanged) The method of claim 9 comprising:

displaying the view in a window of a windowing environment on the video screen.

12. (Unchanged) The method of claim 9 comprising:

providing a template having the hypertext data, the object insertion tag, and a pre-processor directive specifying a parameter; and

processing the template to synthesize the hypertext page from the template, the processing including converting the parameter into hypertext data.

13. (Amended) The method of claim 12 wherein the view of the hypertext page represents a folder of a file system managed by the operating system, the method comprising:

A?

storing a configuration file in the folder specifying the template; and

when a user navigates to the folder, identifying the template to use for synthesizing the hypertext page from the configuration file.

14. (Unchanged) A file system navigation method comprising:

providing a plurality of templates associated with folders in a file system, the templates having an object insertion tag for embedding a folder user interface control;

when a folder is opened by a user, performing the steps of:

processing the template associated with the opened folder to produce a hypertext page having the object insertion tag; and

displaying a view of the hypertext page with the folder user interface control embedded therein, whereby the user manipulates the folder user interface control to activate file system services relating to the folder.

15. (Unchanged) The method of claim 14 comprising:

storing configuration files in at least some of the folders, each configuration file containing data identifying the template associated with the respective folder in which the configuration file is stored;

if the opened folder stores one of the configuration files, determining the template associated with the opened folder from the configuration file stored in the opened folder; and

otherwise, determining a default template is associated with the opened folder.

16. (Unchanged) A hypertext page, comprising:

hypertext data; and

an object insertion tag for embedding a user interface control object, said object providing graphical icon-oriented and/or menu driven user interface elements for controlling operating system and/or file system services when the hypertext page is viewed.

ubB1
A4

17. (Amended) A desktop [in a windowing environment of a graphical user interface] provided by an operating system shell on a computer as a full-screen graphical background display in a windowing environment of a graphical user interface, comprising:

a plurality of graphical icon-oriented and menu-driven user interface elements provided by a control object, the user interface elements being manipulable by a user of the computer to initiate operating system services; and

a view of a hypertext multimedia document incorporating multi-media enhancements and having the control object embedded therein for integrating the multi-media enhancements with the user interface elements, the view constituting at least part of the full screen graphical background display of the desktop in the windowing environment.

REMARKS

Reconsideration of the application is respectfully requested in view of the foregoing amendments and following remarks.

Objection to the Drawings

The drawings have been objected to by the PTO draftsperson for various informalities. As the drawings are acceptable for examination purposes, Applicants will submit corrected and formal drawings to correct these informalities upon allowance.

Objection to the Title

The title has been objected to as not clearly indicative of the claimed invention. Accordingly, Applicants have amended the title to one believed to have more informative value for indexing, classifying or searching.

Alternative Embodiments Support Under § 112, ¶ 1.

Claim 16 has been rejected under 35 U.S.C. § 112, ¶ 1 as lacking support in the specification and drawings. It is well established that the claims as originally filed constitute part of the disclosure of the patent application. Further, "if an application as originally filed contains a claim disclosing material not disclosed in the remainder of the specification, the applicant may amend the specification to include the claimed subject matter." (See, MPEP § 2163.06(c); see also MPEP § 2164, last paragraph.) Accordingly, in the foregoing amendments, Applicants have amended the specification to explicitly incorporate the language from claim 16 into the "Summary" section of the specification. Although Applicants are prepared to cite additional disclosure in the specification in support of the limitations, such citations are seen as unnecessary in view of the amendments. Claim 16 therefore meets the requirements of § 112, ¶ 1.

Patentability Over Gavron.

Claims 1-4 have been rejected under 35 U.S.C. § 102(a) over Gavron. Applicants have amended claim 1, and respectfully traverse the rejection.

Claim 1

Claim 1 is directed towards a system for displaying a graphical user interface to an operating system on a display device of a computer, which includes a desktop displayed as a hypertext multimedia document. In the foregoing amendments, Applicants have amended claim 1 to better clarify what is meant by the desktop in the claimed graphical user interface windowing environment. This amendment clarifies that the desktop is the background display on which the windows are overlaid. In particular, amended claim 1 recites:

> a desktop in the windowing environment of the graphical user interface for providing a view of a hypertext multimedia document as a full-screen graphical background display over which the windows are displayed.

The Office asserts that the original claim language for the desktop element was shown at pp. 102-105, 108 and 109 in Gavron. Applicants respectfully disagree. At pp. 102-105, Gavron shows various "screen shot" views of the Microsoft Windows NT 4 operating system's user interface, where the Microsoft Internet Explorer is run as a maximized window. In this mode of the operating system, the Internet Explorer window is not back-most in the order. Rather, the maximized Internet Explorer window still overlays the desktop, albeit obscuring all but the "task bar" with "start menu" button, open applications buttons and "Tray" with time-of-day of the Windows NT 4 desktop. The Windows NT 4 desktop itself is not a view of a hypertext multimedia document. Pages 108 and 109, on the other hand, show "screen-shot" views of the Microsoft Windows NT 4 operating system desktop overlaid with various dialogs relating to audio. These dialogs are not based on or displayed from hypertext multi-media documents.

Gavron therefore lacks any teaching or suggestion of a desktop that provides a "a view of a hypertext multimedia document as a full-screen graphical background display over which the windows are displayed," as claimed. For at least these reasons, claim 1 and its dependent claims 2-4 should be allowable over this art.

Claims 2-4

Further, claims 2-4 present additional limitations that render these claims separately patentable over this art. For example, claim 2 details that the hypertext multimedia document include icons "operative to launch application programs on user activation." Claim 3 recites, "icons in the hypertext multimedia document... operative to activate file system services." Claim 4 recites that the hypertext multimedia document include a representation "of a folder in a file system of the computer." The hypertext document displays (e.g., within the Internet Explorer window) that are shown in Gavron do not include any such icons with the recited operative functionality or that provide a representation of a file system folder. Claims 2-4 are further allowable over this art for at least these additional reasons.

Applicants note that versions of the Microsoft Windows NT 4 operating system (as well as the Microsoft Windows 98 operating system) subsequent to the filing date of the present application now incorporate a feature called the "active desktop," which resulted from Applicants work and embodies at least some of the claimed invention. Gavron shows the version of the Microsoft Windows NT 4 desktop prior to introduction of the active desktop feature in that operating system.

Patentability over Tittel and Nakajima

Claims 5, 8, 9 and 12 have been rejected under 35 U.S.C. § 103(a) as unpatentable over Tittel in view of Nakajima. Applicants again traverse.

Claim 5

Claim 5 is directed to an operating system shell with a preprocessor that produces hypertext multimedia documents with embedded control objects to provide at least part of an operating system's graphical user interface. In the foregoing amendments, Applicants have amended claim 5 to further clarify the embedded control object. In particular, claim 5 recites,

> a preprocessor for processing the template into a hypertext multimedia document having an embedded control object for providing at least part of an operating system's graphical user interface; and
>
> a viewer for displaying a view of the hypertext document in the graphical user interface wherein the embedded control object is manipulable by a user to activate a service of the operating system. (emphasis added)

The Office asserts that although Tittel lacks the recited preprocessor, it would have been obvious to apply "viewing, processing and embedding modifications" proposed by Nakajima so as to extend operating system shell capabilities. Applicants disagree, and respectfully submit that applying the modifications taught or suggested by Nakajima to Tittel would not result in the claimed invention. Further, the recited preprocessor and embedded control object (that is manipulable to activate an operating system service) are not taught or suggested by either reference, and therefore any combination of the references would continue to lack the recited preprocessor and embedded control object.

Tittel is a basic reference on the Hypertext Markup Language (HTML), which is a hypertext document file format. Tittel describes how HTML is used to produce hypertext multimedia documents for use as "Web pages" that publish information accessible via a browser application from the Internet. (See, e.g., Tittel at cover page ("The fun and easy way to improve your web pages using HTML"); at page 57 ("You'll begin to appreciate what's involved in a markup language and start to understand how to use HTML to create your own Web pages"); at page 388 ("HTML... The SGML-derived markup language used to create Web pages").) Tittel also describes that HTML Web pages are viewed in a browser that runs as an application under an operating system. (See, Tittel at page 15.) Tittel specifically mentions a number of widely available operating systems, including Microsoft Windows 95, Macintosh, DOS, OS/2 and Unix. (See, Tittel at pages 15 and 392.) Tittel lacks any suggestion to use HTML for any other reason than publishing Web pages, and quite significantly lacks any suggestion to use HTML documents as a graphical user interface of an operating system.

Tittel further describes various ways to add multimedia to an HTML Web page, including MIME extensions and tags. (See, Tittel at pages 31-33, 59-60, 76-81 and 101.) Tittel, however, fails to describe or suggest embedding a control object that is manipulable to activate an operating system service into an HTML document. For example, Tittel at page 101 describes an input object tag that "defines an input object within an HTML form." With such an object, a user that browses an HTML form from a Web page can do data entry to submit information to the remote web site, such as a credit card number for on-line shopping. However, Tittel does not describe any control object that is manipulable to activate an operating system service.

Nakajima, on the other hand, describes shell extensions for an operating system. (See, Nakajima at title.) These shell extensions allow application developers to extend the capabilities of a shell and thereby customize the shell to better suit the needs of their users. (See, Nakajima at abstract; and at column 4, lines 56-57.) The shell extensions are implemented as "handlers" that integrate or interact with the operating system shell through a number of system-defined object interfaces (e.g., IShellExtInit, IShellView, IShellBrowser, IShellFolder, etc.). (See, Nakajima at column 87 et seq.) The handlers are essentially plug-in program code modules (i.e., a COM object provided in a DLL) that the application developer designs to implement some extended shell functionality. (See, Nakajima at columns 21-22.) Nakajima describes several specific varieties of handlers, including context menu handlers (which add items to the right-click context menu of a file object); drag-and-drop handlers (which provide some functionality triggered when a user drops and object after dragging to a new position); icon handlers (which add instance-specific icons for file objects or classes of file objects); property-sheet handlers (which add pages to the property sheet dialog box displayed for a file object); and copy-hook handlers (which provide functionality triggered at copy, move, delete or rename of a file object). (See, Nakajima at abstract; at column 4, lines 42-57; and at columns 21-22.) However, Nakajima does not describe or suggest any shell extension handler that displays a view of a hypertext multimedia document either as part of the operating system graphical user interface or otherwise.

The Office asserts that it would have been obvious to add the shell extensions as described by Nakajima to the operating system shell as described in Tittel, as motivated by Nakajima's description that the shell extensions extend "the capabilities provided by a shell of an operating system to allow an application developer to customize [context menus, property sheet pages, icons, drag-and-drop behavior, copy/delete/move/rename behavior, and addition of name spaces visible to the Windows Explorer] ... to better suit the needs of their users." While Applicants agree that modifying an operating system shell to support customization through add-on shell extensions would be useful, Applicants respectfully disagree that such modification would produce the claimed invention. In fact, an operating system that combines all the features identified by the Office from both Tittel and Nakajima was sold by the millions of copies in the fall of 1995, which was the Microsoft Windows 95 operating system. The Microsoft Windows 95 operating system included the Shell Extensions described in Nakajima. (See, Nakajima at columns 21-22.) The Microsoft Windows 95 operating system was also sold with the Microsoft Internet Explorer web browser, which supports browsing of HTML Web pages within an application window running on the Microsoft Windows 95 operating system. (See, Tittel at page 15.) Yet, the combination of Tittel and Nakajima that was fully realized in Microsoft Windows 95 still lacks the preprocessor and embedded control elements recited in claim 5.

Moreover, the shell extensions described by Nakajima merely provide a facility for an application developer to add some particular customizations to the Microsoft Windows 95 operating system shell. Neither Nakajima nor Tittel suggest using the shell extensions to add custom modifications that would integrate hypertext multimedia document views into an operating system graphical user interface. Nakajima teaches that the customizations added via shell extensions can include adding items to the context menus for a file object, modifying drag-and-drop functionality, adding instance- and class-specific file object icons, adding pages to property sheet dialogs of a file object, and modifying copy/move/delete/rename functionality. (See, Nakajima at abstract; at column 4, lines 42-57; and at columns 21-22.) But, there is no statement or suggestion in Nakajima that an application developer could use the shell extensions to add a hypertext multimedia document view to the shell's graphical user interface. Tittel teaches only the use of HTML for Web pages, and thus also would not suggest that HTML could be added to (via shell extensions or otherwise) an operating system shell for displaying parts of an operating system graphical user interface.

Even further, the addition of Nakajima's shell extensions to an operating system shell in and of themselves would not add hypertext multimedia document views into an operating system graphical user interface. At best, the addition of shell extension capability to an operating system is but a starting point to build an operating system shell that displays hypertext multimedia document views as part of the operating system graphical user interface. Considerable additional effort and modification is required. For example, in the illustrated embodiment described in the present application, a web view object 130 was added on top of (in addition to) shell extensions to support display of hypertext multimedia documents as part of the graphical user interface. (See, Specification at page 27, lines 4-17.) In addition, the illustrated embodiment added control objects for access to operating system services (e.g., desktop interface controls 64) that are embeddable into a hypertext multimedia document, and added a preprocessor 60 that generates hypertext multimedia documents (e.g., hypertext page 56) with the desktop interface controls 64 embedded therein from templates 62. Many other modifications made in the illustrated embodiment over the shell extensions of Nakajima are detailed in the Specification.

Accordingly, Nakajima and Tittel (either individually or in combination) utterly lack the recited elements of a preprocessor and an embedded control object manipulable by a user to activate an operating system service, as claimed. Claim 5 (and claims 6-7 that depend therefrom) therefore are clearly allowable over the cited art.

Claim 8

Claim 8 is directed towards a template used to synthesize a hypertext multimedia document. In the foregoing amendment, Applicants have added the language "wherein the embedded control object is manipulable by a user to activate a service of the operating system" to clarify how the document provides an operating system user interface. The claim now recites:

> an object insertion tag for embedding the user interface control object into the hypertext multimedia document, wherein the embedded control object is manipulable by a user to activate a service of the operating system, whereby the preprocessor processes the template into a hypertext multimedia document which provides an operating system user interface when displayed by the viewer.

Tittel and Nakajima fail to teach or suggest the template of claim 8 for at least some of the reasons described above in the discussion of claim 5. Again, the combination of Tittel and Nakajima postulated by the Office would merely result in the Microsoft Windows 95 operating system, which clearly lacks the recited template. None of the various citations from Tittel or Nakajima suggest the many further modifications of the Microsoft Windows 95 operating system that would be required to produce the recited template. In particular, although Tittel describes the use of HTML including the input tag to create Web pages, there is no suggestion in either Tittel or Nakajima of a control object that can be embedded in a hypertext multimedia document and that is manipulable by a user to activate an operating system service. Tittel's input object merely accepts input data for an HTML Web form, and lack any capability to activate operating system services of the local computer. The asserted modification of adding Nakajima's shell extensions to the Tittel operating system would not alter the input object. Accordingly, claim 8 is clearly allowable over this art.

Claim 9

Claim 9 is directed towards a method of providing a graphical user interface to an operating system, that includes generating a view of a hypertext multimedia document with an embedded control "manipulable by a user to activate a service of the operating system." Tittel and Nakajima fail to teach or suggest such a method for at least some of the reasons described above in the discussion of claims 5 and 8. The addition of shell extensions as taught by Nakajima to an operating system that runs a web browser as an application as taught by Tittel would merely result in the Microsoft Windows 95 operating system. Such combination clearly lacks the claimed method which generates and displays a view of a hypertext page with an operating system user interface control. Again, Tittel's input tag and object merely allow data entry into an HTML form, and lack the capability to operate operating system services. The asserted modification of adding Nakajima's shell extensions to the Tittel operating system would not alter the input object. Accordingly, claim 8 is clearly allowable over this art.

Patentability Over Tittel, Gavron and Nakajima

Claims 6, 7, 10, 11 and 13-17 have been rejected under 35 U.S.C. § 103(a) as unpatentable over Tittel in view of Gavron and Nakajima. Applicants traverse the rejection.

Claims 6, 7, 10 and 11

Claim 6 is directed to the operating system shell of claim 5, which further includes a view container object that hosts the viewer and provides a desktop display which has embedded the view of the hypertext multimedia document with embedded control object manipulable to activate an operating system service (e.g., the desktop display depicted in Figure 6 of the present application). Claim 6 has been amended to clarify the desktop display is a full-screen background display to a windowing environment. Amended claim 6 recites, "a view container object for hosting the viewer and providing a desktop display in which the view is embedded as a full-screen graphical background display of a windowing environment."

Claim 7 is directed to an alternative embodiment in which the hypertext multimedia document view having the embedded operating system control object manipulable to activate an operating system service is provided in an application window (e.g., the Windows Explorer display depicted in Figure 7). Claim 7 recites, "a view container object for ... providing an application window in which the view is embedded."

Claims 10 and 11 depend from claim 9, and include limitations respectively similar to claims 6 and 7.

As discussed above for claim 5, Tittel and Nakajima fail to teach or suggest the claimed operating system shell which provides a display with a view of a hypertext multimedia document having an embedded control object manipulable by a user to activate an operating system service. At best, the asserted modification of Tittel to add the shell extensions of Nakajima would result in the Microsoft

Windows 95 operating system, which has Nakajima's shell extensions (that allow an application developer to add on handlers to change context menu items, property sheets, icons, drag-and-drop functionality, and copy/delete/rename functionality) and a Web browser application (Microsoft Internet Explorer) - but, clearly lacks the recited display in its graphical user interface. The further modification asserted by the Office to add the "desktop modifications" introduced to Microsoft Windows NT in the version 4 thereof would not correct this deficiency. In fact, version 4 updates the previous Microsoft Windows NT graphical user interface to more closely resemble that of the Microsoft Windows 95 operating system, so as to incorporate features such as the Briefcase, the Start button, the Taskbar, the Tray, and Windows Explorer browser which are depicted in Gavron at pages 3, 13, and 24-29. This further asserted modification thus would still yield the Microsoft Windows 95 operating system shell and graphical user interface, still lacking the recited display.

Additionally, Applicants note that Gavron clearly teaches that hypertext multimedia documents are viewed in an application window of the Internet Explorer web browser. (See, Gavron at page 107.) The depicted Web page has an input object to enter an Internet search query, and does not activate an operating system service. On the other hand, Gavron shows the various displays generated by the operating system shell, including the desktop and Windows Explorer displays, do not incorporate any views of hypertext multimedia documents. (See, Gavron at pages 3, 13 and 24-29.)

Claims 6, 7, 10 and 11 therefore clearly are allowable over this art.

Claim 13

Claim 13 is directed to the further feature where the template used to produce the hypertext page to represent a folder of the file system is specified in and identified from a configuration file. In particular, claim 13 recites,

storing a configuration file in the folder specifying the template; and
when a user navigates to the folder, identifying the template to use for synthesizing the hypertext page from the configuration file.

The cited art fails to describe or suggest the synthesis of a hypertext page to represent a file system folder from a template, much less specifying in and identifying the template to use for such purpose for a particular folder from a configuration file. The Office asserts that Gavron shows a view of a hypertext page that represents a folder of a file system. Applicants strongly disagree. Gavron shows numerous screen shots of displays that represent the contents of a folder in a file system, such as the Windows Explorer displays shown on pages 28 and 29, the favorites folder display on page 105, the control panel display on page 109, the network neighborhood display on page 155, and even the desktop display on page 3. Not a single one of these displays is generated as a hypertext page. On the other hand, the displays shown in Gavron that are hypertext pages do not represent file folders, such as the Microsoft Web site search page shown at page 100, and the Microsoft Network and Yahoo Web pages shown at page 107. Accordingly, Gavron completely fails to teach or suggest the recited hypertext page that represent a file system folder.

The Office further asserts that Tittel's teaching that "a descriptive markup language describes the structure and behavior of a document" implicitly shows "storing a configuration file in the folder specifying a template [to use for synthesizing the hypertext page that represents the folder]." Not so. The hypertext markup language taught by Tittel describes the structure and behavior of a document. This would not suggest where or how to store information to specify a template from which a representation of a particular file folder is to be synthesized.

Claim 13 therefore clearly is allowable over the cited art.

Claim 14

Claim 14 is directed to a file system navigation method in which a view of a hypertext page with an embedded control to activate file system services relating to a folder is displayed when the folder is opened. In particular, claim 14 recites:

> providing a plurality of templates associated with folders in a file system, the templates having an object insertion tag for embedding a folder user interface control;
>
> when a folder is opened by a user, performing the steps of:
>
> processing the template associated with the opened folder to produce a hypertext page having the object insertion tag; and
>
> displaying a view of the hypertext page with the folder user interface control embedded therein, whereby the user manipulates the folder user interface control to activate file system services relating to the folder.

As discussed above, the asserted combination of Tittel, Gavron and Nakajima would have resulted, at best, in the shell and graphical user interface of the Microsoft Windows 95 operating system. This combination clearly lacks the recited operation of the claimed file system navigation method, which displays a view of a hypertext page with an embedded control to activate file system services relating to a folder when the folder is opened. There is no statement in any of these references that would be taken by one of ordinary skill in the art to further modify the Microsoft Windows 95 operating system to provide this recited operation.

The Office cites to Gavron as showing "the view of the hypertext page represents a folder of a file system." No such teaching is made by Gavron. As discussed above in connection with claim 13, Gavron clearly shows the graphical user interface displays that represent file system folders in the Microsoft Windows NT 4 Workstation operating system are <u>not</u> hypertext pages. Whereas, all the hypertext pages that are shown in Gavron are Web pages that do <u>not</u> represent file system folders. Accordingly, no such suggestion can be taken from Gavron. Tittel and Nakajima also lack any such suggestion.

Moreover, the cited art fails to teach or suggest the folder user interface control embedded in the hypertext page via an object insertion tag, which the user manipulates to activate file system services relating to the folder. As already discussed, Tittel describes a tag and an input object for data entry into an HTML Web page form, and lacks any suggestion to embed a folder user interface control that actives file system services on a hypertext page. (See, Tittel at page 101.) Nakajima at columns 13 and 14 describes use of OLE embedding for drag-and-drop as well as name space extensions, and thus would have lead towards use of OLE embedding for a folder user interface control and away from object insertion tags and hypertext page displays for such controls. Gavron shows an operating system shell which places folder user interface control icons on graphical user interface displays that are not hypertext pages, and thus also would lead away from embedding such controls on a hypertext page via an object insertion tag.

Claim 14 therefore clearly is allowable over this art.

Claim 15

Claim 15 depends from claim 14 and adds limitations similar in some respects to those recited in claim 13. Claim 15 is allowable over the cited art for at least some of the reasons discussed above as for claim 13.

Claim 16

Claim 16 is directed towards a hypertext page that has an embedded user interface control object to control operating system and/or file system services. In particular, claim 16 recites, "an object insertion tag for embedding a user interface control object, said object providing graphical icon-oriented and/or menu driven user interface elements for controlling operating system and/or file system services when the hypertext page is viewed."

As discussed previously, the cited art fails to teach or suggest embedding a user interface control object onto a hypertext page. Tittel describes a tag and input object for data entry into an HTML Web page form, and lacks any suggestion to embed a folder user interface control that actives file system services on a hypertext page. (See, Tittel at page 101.) This deficiency is not made up by either of the other references, which in fact explicitly teach away. Nakajima at columns 13 and 14 describes use of OLE embedding for drag-and-drop as well as name space extensions, and thus would have lead towards use of OLE embedding of a user interface control into the shell and away from object insertion tags and hypertext page displays for such controls. Gavron shows placing such user interface controls on graphical user interface displays that are not hypertext pages, which also leads away from embedding such controls on a hypertext page via an object insertion tag.

Accordingly, Claim 16 clearly is allowable over this art.

Claim 17

Claim 17 is directed towards a desktop that includes a view of a hypertext multimedia document that incorporates multi-media enhancements and has an embedded control object to provide graphical icon-oriented and menu-driven user interface elements manipulable by a user to initiate operating system services. In particular, claim 17 recites,

> A desktop provided by an operating system shell on a computer as a full-screen graphical background display in a windowing environment of a graphical user interface, comprising:
>
> a plurality of graphical icon-oriented and menu-driven user interface elements provided by a control object, the user interface elements being manipulable by a user of the computer to initiate operating system services; and
>
> a view of a hypertext multimedia document incorporating multi-media enhancements and having the control object embedded therein for integrating the multi-media enhancements with the user interface elements, the view constituting at least part of the full screen graphical background display of the desktop in the windowing environment.

As discussed above, the cited references fail to teach or suggest the display of a hypertext multimedia document as the desktop in a windowing environment of a graphical user interface, and also fail to teach or suggest such hypertext page have an embedded control object to provide graphical icon-oriented and menu-driven user interface elements manipulable by a user to initiate operating system services.

The Office asserts that it would have been obvious to apply the icon and windowing modifications shown in Gavron to the operating system shell shown in Tittel so as to provide an approach to setting up the desktop to look and feel the way you like. Tittel describes running a Internet Web browser, such as the Microsoft Internet Explorer (or Netscape Navigator), on an operating system, such as the Microsoft Windows 95 (or Unix, DOS, OS/2, or Mac). On the other hand, Gavron describes graphical user interface features of the Microsoft Windows NT 4 Workstation operating system. These features update the Windows NT Workstation graphical user interface to resemble that of the Microsoft Windows 95 operating system. It is therefore not understood how applying any feature taught in Gavron would in any way alter the Microsoft Windows 95 operating system's user interface with the Microsoft Internet Explorer run thereon.

The Office also asserts it would have been obvious to apply OLE embedding in the shell and user interface of the operating system to the Gavron and Tittel combination, so as to extend shell capabilities to allow application developer customizations that better suit user needs. Again, it is hard to understand how this proposed modification would lead toward the claimed invention. OLE embedding already was incorporated into the Microsoft Windows 95 operating system (and the Microsoft Windows NT 4 Workstation operating system), and therefore its application to the Gavron and Tittel combination would effect no change. Further, to the extent that OLE embedding already provides a way in these operating system shells to incorporate user interface control elements in graphical user interface displays of the operating system shell, any such suggestion to use OLE embedding actually would teach away from the claimed invention.

Accordingly, claim 17 clearly is allowable over the cited art.

Conclusion

The prior art made of record but not relied upon by the Examiner is believed to be no more pertinent to applicant's invention than the cited references for the reasons given above.

The claims in their present form should now be allowable. Such action is respectfully requested.

Respectfully submitted,
KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP

By
Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446
cc: Patent Group Docket Dept.
Megan Thomas

08/761,699

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

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR		ATTORNEY DOCKET NO.
08/761,699	12/06/96	SLIVKA	B	3382-46311

#8/B

LM71/1027

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON
ONE WORLD TRADE CENTER SUITE 1600
121 SW SALMON STREET
PORTLAND OR 97204-2988

EXAMINER
YOUNG, J

ART UNIT	PAPER NUMBER
2754	8

DATE MAILED: 10/27/99

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

Commissioner of Patents and Trademarks

PTO-90C (Rev. 2/95)

1- File Copy

Notice of Allowability	Application No. 08/761,699	Applicant(s) Slivka et al.
	Examiner John L. Young	Group Art Unit 2764

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance and Issue Fee Due or other appropriate communication will be mailed in due course.

☒ This communication is responsive to *Amendment A filed 7/27/99 as paper # 7*.

☒ The allowed claim(s) is/are *1-17*.

□ The drawings filed on ________ are acceptable.

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

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

□ received.

□ received in Application No. (Series Code/Serial Number) ________.

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

*Certified copies not received: ________.

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

A SHORTENED STATUTORY PERIOD FOR RESPONSE to comply with the requirements noted below is set to EXPIRE **THREE MONTHS** FROM THE "DATE MAILED" of this Office action. Failure to timely comply will result in ABANDONMENT of this application. Extensions of time may be obtained under the provisions of 37 CFR 1.136(a).

□ Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL APPLICATION, PTO-152, which discloses that the oath or declaration is deficient. A SUBSTITUTE OATH OR DECLARATION IS REQUIRED.

☒ Applicant MUST submit NEW FORMAL DRAWINGS

□ because the originally filed drawings were declared by applicant to be informal.

☒ including changes required by the Notice of Draftsperson's Patent Drawing Review, PTO-948, attached hereto or to Paper No. 5.

□ including changes required by the proposed drawing correction filed on ________, which has been approved by the examiner.

□ including changes required by the attached Examiner's Amendment/Comment.

Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the reverse side of the drawings. The drawings should be filed as a separate paper with a transmittal lettter addressed to the Official Draftsperson.

□ Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Any response to this letter should include, in the upper right hand corner, the APPLICATION NUMBER (SERIES CODE/SERIAL NUMBER). If applicant has received a Notice of Allowance and Issue Fee Due, the ISSUE BATCH NUMBER and DATE of the NOTICE OF ALLOWANCE should also be included.

Attachment(s)

☒ Notice of References Cited, PTO-892

□ Information Disclosure Statement(s), PTO-1449, Paper No(s). ________

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

□ Notice of Informal Patent Application, PTO-152

□ Interview Summary, PTO-413

☒ Examiner's Amendment/Comment

□ Examiner's Comment Regarding Requirement for Deposit of Biological Material

☒ Examiner's Statement of Reasons for Allowance

NOTICE OF ALLOWABILITY

DRAWINGS

1. This application was submitted with informal drawings. "The applicant is required to submit acceptable drawings within three months from the mailing of the 'Notice of Allowability.'" (See 37 CFR 1.85(c)).

TITLE OBJECTION

2. **Obejection Withdrawn.**

CLAIM REJECTIONS — 35 U.S.C. §112 ¶ 1

3. **Rejections Withdrawn.**

CLAIM REJECTIONS — 35 U.S.C. §102

4. **Rejections Withdrawn.**

CLAIM REJECTIONS — 35 U.S.C. §103(a)

5. **Rejections Withdrawn.**

EXAMINER'S FORMAL AMENDMENT

An examiner's amendment to the record appears below. Should the changes and/or additions be unacceptable to applicant, an amendment may be filed as provided by 37 CFR 1.312. To ensure consideration of such an amendment, it MUST be submitted no later than the payment of the issue fee.

Authorization for this examiner's amendment was given in a telephone interview with Stephen A. Wight (attorney for applicant) on 10/25/99.

This application was initially filed with claims 1-17. Amendment A amended the title and claims 1, 5-10, 13 & 17. The title of the invention and Claim 17 are herein amended by examiner's amendment.

Amendment to the Title

6. The following change in title is herein incorporated in the notice of allowability:

OPERATING SYSTEM SHELL HAVING A WINDOWING GRAPHICAL USER INTERFACE WITH A DESKTOP DISPLAYED AS A HYPERTEXT MULTIMEDIA DOCUMENT.

This change in title "may result in [a] slightly longer [title], but the loss in brevity of title will be more than offset by the gain in its informative value in indexing, classifying, searching, etc." (See MPEP 606.01).

Amendment to Independent Claim 17

7. The following change in claim 17 is herein incorporated in the notice of allowability:

17. (Twice Amended) A desktop provided by an operating system shell on a computer as a full-screen graphical background display of a hypertext multimedia document in a windowing environment of a graphical user interface, comprising:

a plurality of graphical icon-oriented and menu-driven user interface elements provided by a control object, the user interface elements being manipulable by a user of the computer to initiate operating system services; and

a view of a hypertext multimedia document incorporating multi-media enhancements and having the control object embedded therein for integrating the multi-media enhancements with the user interface elements, the view constituting at least part of the full screen graphical background display of the desktop in the windowing environment.

CLAIMS ALLOWABLE OVER THE PRIOR ART

8. **Claims 1-17 are allowable.**

REASONS FOR ALLOWABILITY

9. The following is an examiner's statement of reasons for allowability of claims 1-17: The applicant has sufficiently shown that limitations of the instant application have multiple implementation details not disclosed in the prior art of record; for example:

Independent claim 1 as amended substantially recites: "a desktop in the windowing environment of the graphical user interface for providing a view of a hypertext multimedia document as a full-screen graphical background display over which the windows are displayed." Support for this limitation may be found in the specification of the instant application at: p. 9, ll. 8-12. Even though, hypertext multimedia documents and desktops were independently known in the art, the prior art references of record do not precisely teach or suggest a desktop which itself is a hypertext multimedia document.

This hypertext desktop is distinguished over the prior art of record; for example, the Gavron reference shows a hypertext document overlaid on a desktop (as opposed to showing a hypertext document which is per se the desktop configuration).

Therefore, the instant invention is not anticipated by the prior art of record. Furthermore, the prior art of record does not implicitly, individually or in combination disclose elements that would render the claimed limitations in the instant invention obvious to one of ordinary skill in the art.

Dependent claims 2-4 are allowable because they depend from claim 1 which contains allowable subject matter.

Independent claim 5 as amended substantially recites: "a viewer for displaying a view of the hypertext document in the graphical user interface wherein the embedded control object is manipulable by a user to activate a service of the operating system." Support for this limitation may be found in the specification of the instant application at: p. 30, ll.33-36; p. 31, ll. 1-34; p. 32, ll. 1-34; and p.33, ll. 1-22. Even though object linking and embedding (OLE) techniques were known in the art, the prior art references of record do not precisely teach or suggest "a viewer for displaying a view of the hypertext document in the graphical user interface wherein the embedded control object is manipulable by a user to activate a service of the operating system."

Therefore, the instant invention is not anticipated by the prior art of record. Furthermore, the prior art of record does not implicitly, individually or in combination disclose elements that would render the claimed limitations in the instant invention obvious to one of ordinary skill in the art.

Dependent claims 6-7 are allowable because they depend from claim 5 which contains allowable subject matter.

Independent claims 8 & 9 are allowable for substantially the same reasons as claim 5, i.e., the prior art references of record do not precisely teach or suggest "the embedded control object is manipulable by a user to activate a service of the operating system."

Dependent claims 10-13 are allowable because they depend from base claims that contain allowable subject matter.

Independent claim 14 substantially recites: "templates having an object insertion tag for embedding a folder user interface control. . . ." the prior art references of record do not show object insertion tags and hypertext page displays for such controls; therefore, claim 14 is allowable over the prior art of record. Therefore, the instant invention is not anticipated by the prior art of record. Furthermore, the prior art of record does not implicitly, individually or in combination disclose elements that would render the claimed limitations in the instant invention obvious to one of ordinary skill in the art.

Dependent claim 15 is allowable because it depends from claim 14 which contains allowable subject matter.

Independent claim 16 is allowable for substantially the same reasons as claim 14, i.e., the prior art references of record do not precisely teach or suggest an "object insertion tag for embedding a . . . user interface control object. . . ."

Independent claim 17 is allowable for substantially the same reasons as claim 1, i.e., the prior art references of record do not precisely teach or suggest a desktop which itself is a hypertext multimedia document.

RELEVANT REFERENCES

10. The references made of record herein and not relied upon are considered pertinent to Applicants' disclosure:

5,877,765, U.S. Pat. [Mar. 2, 1999] Dickman et al., 345/349
"METHOD AND SYSTEM FOR DISPLAYING INTERNET SHORTCUT ICONS ON THE DESKTOP." This reference discusses a hyperlinked desktop.

5,905,492, U.S. Pat. [May 18, 1999] Straub et al., 345/333
"DYNAMICALLY UPDATING THEMES FOR AN OPERATING SYSTEM SHELL." This reference discusses an operating system shell with a hypertext desktop.

CONCLUSION

11. Any response to this action should be mailed to:

Commissioner of Patents and Trademarks
Washington, D.C. 20231

Any response to this action may be sent via facsimile to either:

(703) 308-9051 (for formal communications marked EXPEDITED PROCEDURE), or

(703) 308-5397 (for informal communications marked PROPOSED or DRAFT).

Hand delivered responses may be brought to:

Sixth floor Receptionist
Crystal Park II
2121 Crystal Drive
Arlington, Virginia.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to John L. Young who may be reached via telephone at (703) 305-3801. The examiner can normally be reached Monday through Friday between 8:30 A.M. and 5:00 P.M.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, James Trammell, may be reached at (703) 305-9768.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the Group receptionist whose telephone number is (703) 305-3900.

John L. Young

Patent Examiner

October 26, 1999

James P. Trammell
Supervisory Patent Examiner
Technology Center 2700

Notice (,ferences Cited	Application No. 08/761,699	Applicant(s ˆlivka et al.	
	Examiner John L. Young	Group Art Unit 2764	Page 1 of 1

U.S. PATENT DOCUMENTS

*		DOCUMENT NO.	DATE	NAME	CLASS	SUBCLASS
x	A	5,831,606	11/3/98	Nakajima et al.,	345	326
x	B	5,905,492	5/18/99	Straub et al.,	345	333
x	C	5,877,765	3/2/99	Dickman et al.,	345	349
	D					
	E					
	F					
	G					
	H					
	I					
	J					
	K					
	L					
	M					

FOREIGN PATENT DOCUMENTS

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

NON-PATENT DOCUMENTS

*		DOCUMENT (Including Author, Title, Source, and Pertinent Pages)	DATE
x	U	Gavron, Jacquelyn and Joseph Moran. "How to Use Microsoft (R) Windows (R) NT 4 WORKSTATION." (Emeryville: Ziff Davis Press, 1996) pp. 24, 25, 28, 29, 88, 89, 100-105, 108, 109, 154-157 & 167.	1/1/96
x	V	Tittle, Ed. and Steve James. "HTML FOR DUMMIES" 2d ed. (Foster City: IDG Books Worldwide, 1996) pp. 14, 15, 31-37, 57, 58, 60, 76-81, 131-133, 158, 350, 351, 385, 388, 392, 396 & 397.	3/11/96
	W		
	X		

* A copy of this reference is not being furnished with this Office action.
(See Manual of Patent Examining Procedure, Section 707.05(a).)

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office

NOTICE OF ALLOWANCE AND ISSUE FEE DUE

LM21-1022

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON
ONE WORLD TRADE CENTER SUITE 1600
121 SW SALMON STREET
PORTLAND OR 97204-2988

APPLICATION NO.	FILING DATE	TOTAL CLAIMS	EXAMINER AND GROUP ART UNIT		DATE MAILED
08/761,709	12/06/96	057	YOUNG, J	2764	10/22/99

First Named Applicant: SLIVKA, 35 USC 154(b) term ext. = 0 Days.

TITLE OF INVENTION: OPERATING SYSTEM SHELL HAVING A WINDOWING GRAPHICAL USER INTERFACE WITH A DESKTOP DISPLAYED AS A HYPERTEXT MULTIMEDIA DOCUMENT (AS AMENDED)

ATTY'S DOCKET NO.	CLASS-SUBCLASS	BATCH NO.	APPLN. TYPE	SMALL ENTITY	FEE DUE	DATE DUE
2 3382-45408	707-513.000	U52	UTILITY	NO	$1210.00	01/22/00

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. <u>PROSECUTION ON THE MERITS IS CLOSED.</u>

THE ISSUE FEE MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED.

HOW TO RESPOND TO THIS NOTICE:

I. Review the SMALL ENTITY status shown above.
If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

A. If the status is changed, pay twice the amount of the FEE DUE shown above and notify the Patent and Trademark Office of the change in status, or
B. If the status is the same, pay the FEE DUE shown above.

If the SMALL ENTITY is shown as NO:

A. Pay FEE DUE shown above, or

B. File verified statement of Small Entity Status before, or with, payment of 1/2 the FEE DUE shown above.

II. Part B-Issue Fee Transmittal should be completed and returned to the Patent and Trademark Office (PTO) with your ISSUE FEE. Even if the ISSUE FEE has already been paid by charge to deposit account, Part B Issue Fee Transmittal should be completed and returned. If you are charging the ISSUE FEE to your deposit account, section "4b" of Part B-Issue Fee Transmittal should be completed and an extra copy of the form should be submitted.

III. All communications regarding this application must give application number and batch number.
Please direct all communications prior to issuance to Box ISSUE FEE unless advised to the contrary.

***IMPORTANT REMINDER:* Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.**

PATENT AND TRADEMARK OFFICE COPY

PTOL-85 (REV. 10-96) Approved for use through 06/30/99. (0651-0033)

*U.S. GPO: 1999-454-457/24601

PART B—ISSUE FEE TRANSMITTAL

Complete and mail this form, together with appl. ,le fees, to: **Box ISSUE FEE**
Assistant Commissioner for Patents
Washington, D.C. 20231

MAILING INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE. Blocks 1 through 4 should be completed where appropriate. All further correspondence including the Issue Fee Receipt, the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Legibly mark-up with any corrections or use Block 1)

LM71/1027

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON
ONE WORLD TRADE CENTER SUITE 1600
121 SW SALMON STREET
PORTLAND OR 97204-2988

Note: The certificate of mailing below can only be used for domestic mailings of the Issue Fee Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing.

Certificate of Mailing

I hereby certify that this Issue Fee Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Box Issue Fee address above on the date indicated below.

Stephen A. Wight (Depositor's name)

(Signature)

12-17-99 (Date)

APPLICATION NO.	FILING DATE	TOTAL CLAIMS	EXAMINER AND GROUP ART UNIT		DATE MAILED
08/761,699	12/06/96	017	YOUNG, J	2764	10/27/99

First Named Applicant: SLIVKA, 35 USC 154(b) term ext. = 0 Days.

TITLE OF INVENTION: OPERATING SYSTEM SHELL HAVING A WINDOWING GRAPHICAL USER INTERFACE WITH A DESKTOP DISPLAYED AS A HYPERTEXT MULTIMEDIA DOCUMENT (AS AMENDED)

ATTY'S DOCKET NO.	CLASS-SUBCLASS	BATCH NO.	APPLN. TYPE	SMALL ENTITY	FEE DUE	DATE DUE
2 3382-45418	707-513.000	D57	UTILITY	NO	$1210.00	01/27/00

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363). Use of PTO form(s) and Customer Number are recommended, but not required.

□ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.

□ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47) attached.

2. For printing on the patent front page, list (1) the names of up to 3 registered patent attorneys or agents OR, alternatively, (2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.

1 KLARQUIST SPARKMAN
2 CAMPBELL LEIGH &
3 WHINSTON, LLP

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)
PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. Inclusion of assignee data is only appropiate when an assignment has been previously submitted to the PTO or is being submitted under separate cover. Completion of this form is NOT a subsititue for filing an assignment.

(A) NAME OF ASSIGNEE Microsoft Corporation

(B) RESIDENCE: (CITY & STATE OR COUNTRY) Redmond, Washington

Please check the appropriate assignee category indicated below (will not be printed on the patent)

□ individual ☒ corporation or other private group entity □ government

4a. The following fees are enclosed (make check payable to Commissioner of Patents and Trademarks):

☒ Issue Fee
☒ Advance Order - # of Copies fourteen (14)

4b. The following fees or deficiency in these fees should be charged to:

DEPOSIT ACCOUNT NUMBER 02-4550
(ENCLOSE AN EXTRA COPY OF THIS FORM)

□ Issue Fee
□ Advance Order - # of Copies ______

The COMMISSIONER OF PATENTS AND TRADEMARKS IS requested to apply the Issue Fee to the application identified above.

(Authorized Signature) (Date) 12-17-99

NOTE: The Issue Fee will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the Patent and Trademark Office.

12/22/1999 STEFERR1 00000010 08761699

01 FC:142	1210.00 OP
02 FC:561	42.00 OP

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending on the needs of the individual case. Any comments on the amount of time required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, D.C. 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND FEES AND THIS FORM TO: Box Issue Fee, Assistant Commissioner for Patents, Washington D.C. 20231

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

TRANSMIT THIS FORM WITH FEE

PTOL-85B (REV.10-96) Approved for use through 06/30/99. OMB 0651-0033

Patent and Trademark Office; U.S. DEPARTMENT OF COMMERC

SAW:mlt 12/17/99 3382-45418

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

Slivka et al.

Application No.: 08/761,699

Filed: December 6, 1996

For: OPERATING SYSTEM SHELL HAVING A WINDOWING GRAPHICAL USER INTERFACE WITH A DESKTOP DISPLAYED AS A HYPERTEXT MULTIMEDIA DOCUMENT

Examiner: J. Young

Date: December 17, 1999

Art Unit: 2764

Batch No: D57

CERTIFICATE OF MAILING

I hereby certify that this paper and the documents referred to as being attached or enclosed herewith are being deposited with the United States Postal Service on December 17, 1999 as First Class Mail in an envelope addressed to: BOX ISSUE FEE, ASSISTANT COMMISSIONER FOR PATENTS, WASHINGTON D.C. 20231.

Attorney for Applicant

LETTER TO THE OFFICIAL DRAFTSPERSON

BOX ISSUE FEE
TO THE ASSISTANT COMMISSIONER FOR PATENTS
Washington, DC 20231

In response to the NOTICE OF ALLOWABILITY mailed on October 27, 1999, attached please find 3 sheets of formal drawings for this application, with each sheet indicating the Application Number and Art Unit on the reverse side, together with a copy of the Notice of Allowability.

[X] Please return the enclosed postcard to confirm that the items listed above have been received.

Respectfully submitted,

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP

By
Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446
cc: Patent Group Docketing Dept. (82104)
KS Paralegal

Notice of Allowability	Application No. 08/761,699	Applicant(s) Slivka et al.	
	Examiner John L. Young	Group Art Unit 2764	

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance and Issue Fee Due or other appropriate communication will be mailed in due course.

☒ This communication is responsive to *Amendment A filed 7/27/99 as paper # 7* .

☒ The allowed claim(s) is/are *1-17* .

□ The drawings filed on ______________ are acceptable.

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

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

□ received.

□ received in Application No. (Series Code/Serial Number) ______________ .

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

*Certified copies not received: ______________ .

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

A SHORTENED STATUTORY PERIOD FOR RESPONSE to comply with the requirements noted below is set to EXPIRE THREE MONTHS FROM THE "DATE MAILED" of this Office action. Failure to timely comply will result in ABANDONMENT of this application. Extensions of time may be obtained under the provisions of 37 CFR 1.136(a).

□ Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL APPLICATION, PTO-152, which discloses that the oath or declaration is deficient. A SUBSTITUTE OATH OR DECLARATION IS REQUIRED.

☒ Applicant MUST submit NEW FORMAL DRAWINGS

□ because the originally filed drawings were declared by applicant to be informal.

☒ including changes required by the Notice of Draftsperson's Patent Drawing Review, PTO-948, attached hereto or to Paper No. *5* .

□ including changes required by the proposed drawing correction filed on ______________ , which has been approved by the examiner.

□ including changes required by the attached Examiner's Amendment/Comment.

Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the reverse side of the drawings. The drawings should be filed as a separate paper with a transmittal lettter addressed to the Official Draftsperson.

□ Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Any response to this letter should include, in the upper right hand corner, the APPLICATION NUMBER (SERIES CODE/SERIAL NUMBER). If applicant has received a Notice of Allowance and Issue Fee Due, the ISSUE BATCH NUMBER and DATE of the NOTICE OF ALLOWANCE should also be included.

Attachment(s)

☒ Notice of References Cited, PTO-892

□ Information Disclosure Statement(s), PTO-1449, Paper No(s). ________

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

□ Notice of Informal Patent Application, PTO-152

□ Interview Summary, PTO-413

☒ Examiner's Amendment/Comment

□ Examiner's Comment Regarding Requirement for Deposit of Biological Material

☒ Examiner's Statement of Reasons for Allowance

SAW:mlt 12/17/99 3382-45418

PATENT
Attorney's Matter No. 3382-45418

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

B.$

In re application of: Slivka et al.

Application No.: 08/761,699

Filed: December 6, 1996

For: OPERATING SYSTEM SHELL HAVING A WINDOWING GRAPHICAL USER INTERFACE WITH A DESKTOP DISPLAYED AS A HYPERTEXT MULTIMEDIA DOCUMENT

Examiner: J. Young

Date: December 17, 1999

Art Unit: 2764

Batch No. D57

CERTIFICATE OF MAILING

I hereby certify that this paper and the documents referred to as being attached or enclosed herewith are being deposited with the United States Postal Service on December 17, 1999 as First Class Mail in an envelope addressed to: BOX ISSUE FEE, ASSISTANT COMMISSIONER FOR PATENTS, WASHINGTON, D.C. 20231.

Stephen A. Wight

Attorney for Applicant

TRANSMITTAL LETTER

BOX ISSUE FEE
TO THE ASSISTANT COMMISSIONER FOR PATENTS
Washington, DC 20231

Enclosed for filing in the above-referenced application are the following:

☒ In connection with issuance of a patent:
  ☒ Form PTOL-85
☒ Advance order of 14 copies (Fee $42.00)
☒ Issue Fee ($1210.00)
☒ A check in the amount of $1252.00 to cover the above-listed fees.
☒ The Commissioner is hereby authorized to charge any additional fees which may be required in connection with issuance of a patent, or credit overpayment, to Account No. 02-4550. A copy of this sheet is enclosed.
☒ Please return the enclosed postcard to confirm that the items listed above have been received.

Respectfully submitted,

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP

By Stephen A. Wight
Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446
cc: Patent Group Docketing Dept. (82104.1)
KS Paralegal

SAW:mlt 12/17/99 3382-45418

PATENT
Attorney's Matter No. 3382-45418

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Slivka et al.

Application No.: 08/761,699

Filed: December 6, 1996

For: OPERATING SYSTEM SHELL HAVING A WINDOWING GRAPHICAL USER INTERFACE WITH A DESKTOP DISPLAYED AS A HYPERTEXT MULTIMEDIA DOCUMENT

Examiner: J. Young

Date: December 17, 1999

Art Unit: 2764

Batch No. D57

CERTIFICATE OF MAILING

I hereby certify that this paper and the documents referred to as being attached or enclosed herewith are being deposited with the United States Postal Service on December 17, 1999 as First Class Mail in an envelope addressed to: BOX ISSUE FEE, ASSISTANT COMMISSIONER FOR PATENTS, WASHINGTON, D.C. 20231.

Attorney for Applicant

TRANSMITTAL LETTER

BOX ISSUE FEE
TO THE ASSISTANT COMMISSIONER FOR PATENTS
Washington, DC 20231

Enclosed for filing in the above-referenced application are the following:

☒ In connection with issuance of a patent:
  ☒ Form PTOL-85
☒ Advance order of 14 copies (Fee $42.00)
☒ Issue Fee ($1210.00)
☒ A check in the amount of $1252.00 to cover the above-listed fees.
☒ The Commissioner is hereby authorized to charge any additional fees which may be required in connection with issuance of a patent, or credit overpayment, to Account No. 02-4550. A copy of this sheet is enclosed.
☒ Please return the enclosed postcard to confirm that the items listed above have been received.

Respectfully submitted,

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP

By
Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446
cc: Patent Group Docketing Dept. (82104.1)
KS Paralegal

```
     (FILE 'USPAT' ENTERED AT 16:37:58 ON 15 MAR 1999)
L1            0 S (HYPERTEXT OR HTML) (5A) MULTIMEDIA (5A) DOCUMENT? (5A)
TEM
L2            0 S (HYPERTEXT OR HTML) (10A) MULTIMEDIA (10A) DOCUMENT? (10
A)
L3            0 S (HYPERTEXT OR HTML) (5A) MULTIMEDIA (5A) TEMPLATE?
L4            6 S (HYPERTEXT OR HTML) (5A) DOCUMENT? (5A) TEMPLATE?
L5            2 S L4 AND MULTIMEDIA
L6            0 S L5 AND OPERATING SYSTEM (3A) SHELL
L7            0 S L4 AND OPERATING SYSTEM (3A) SHELL
L8           23 S OPERATING SYSTEM SHELL
L9            0 S L8 AND L5
L10           0 S L8 AND L4
L11           2 S L8 AND (HTML OR HYPERTEXT)
L12           0 S L11 AND TEMPLATE?
L13           0 S L11 AND TEMPLATE?
L14           2 S L5 AND TEMPLATE?
L15           6 S L4 AND TEMPLATE?
L16           0 S L14 AND OPERATING SYSTEM SHELL
L17           0 S L14 AND OPERATING SYSTEM (5A) COMMAND? (5A) INTERPRET?
L18           0 S L14 AND OPERATING SYSTEM (5A) COMMAND?
L19           0 S L14 AND OPERATING SYSTEM (5A) INTERPRET?
L20           0 S L4 AND OPERATING SYSTEM (3A) INTERPRET?
L21           6 S L15 AND PROCESSOR?
L22           2 S L14 AND PROCESSOR?
L23         129 S TEMPLATE (3A) PROCESSOR?
L24           1 S L23 AND HTML
L25           9 S L8 AND VIEWER?
L26           0 S L25 AND HTML
L27           0 S L25 AND HYPER
L28           8 S L25 AND DOCUMENT?
L29           0 S L8 AND EMBED? CONTROL OBJECT
L30           0 S L8 AND EMBED?(3A) CONTROL? (3A) OBJECT?
```

```
d his

     (FILE 'USPAT' ENTERED AT 10:05:44 ON 16 MAR 1999)
L1              0 S OPERATING (3A) SYSTEM (3A) SHELL (300A) EMBED? (5A)CONTR
OL
L2              9 S OPERATING (3A) SYSTEM (3A) SHELL (300A) CONTROL (3A) OBJ
ECT
L3              7 S L2 AND EMBED?
L4              7 S L3 AND (GUI OR USER (3A) INTERFACE?)
L5              7 S L4  AND VIEWER?
L6              0 S L5 AND TEMPLATE? (3A) PROCESSOR?
L7              7 S L5 AND PROCESSOR?
L8             62 S TEMPLATE? (1A) PROCESSOR?
L9              1 S L8 AND (HTML OR HYPER)
```

3-16-99

```
Please ENTER, separated by commas, the NAMES of the files you want to search.
You may select as many files as you want, including files that do not appear
below, but you must enter them all at one time.  To see a description of a
file, ENTER its page (PG) number.
            FILES - PAGE 1 of 10 (NEXT PAGE for additional files)

 NAME   PG DESCRIP          NAME   PG DESCRIP         NAME   PG DESCRIP

   -- C O M P U T E R S  &  C O M M U N I C A T I O N S  L I B R A R Y --

  ---- Full-Text News ---    ------ Forecasts ------      --- Assists ---
 CURNWS  1 Last 2 years     CMPIND  8 Cmptr Forecasts GUIDE   1 Descriptions *
 ARCNWS  1 Beyond 2 years
 ALLNWS  1 Comp/Comm News

 - Full-Text News By Topic -
 COMM    1 Commun News
 CMPTRS  1 Computer News
 ELTRNC  1 Electronic News
 TECHNY  1 Technology News

SEE NEXT PAGE FOR ADDITIONAL SOURCES.  Files marked  *  may not be combined.
```

```
Please ENTER, separated by commas, the NAMES of the files you want to search.
You may select as many files as you want, including files that do not appear
below, but you must enter them all at one time.  To see a description of a
file, ENTER its page (PG) number.
            FILES - PAGE 2 of 10 (NEXT PAGE or PREV PAGE for additional files)

 NAME   PG DESCRIP                      NAME   PG DESCRIP

 ------------------------------- Full-Text News -------------------------------
 ATM     2 Advanced TV Mkts             CSERP   2 Cable Satellite Europe
 AMNEWS  2 AM Newswire                  CSEXP   2 Cable Satellite Express
 ATM2    2 ATM - The Magazine           CBFXDY  2 Cablefax Daily
 BATTEV  2 Battery & EV Tech            CADCAM  2 CAD/CAM Update
 BILWLD  2 Billing World                CDNEWS  2 CD Computing News
 BRDBND  2 BroadBand Net Nws            CDDATA  2 CD-ROM Databases
 BYTE    2 BYTE                         COMDLY  2 Communications Daily
 CBINT   2 Cable Bus Int                COMTDY  2 Communic. Today
 CSASIA  2 Cable Satellite Asia

SEE NEXT PAGE FOR ADDITIONAL SOURCES.
```

```
Please ENTER, separated by commas, the NAMES of the files you want to search.
You may select as many files as you want, including files that do not appear
below, but you must enter them all at one time.  To see a description of a
file, ENTER its page (PG) number.
            FILES - PAGE 3 of 10 (NEXT PAGE or PREV PAGE for additional files)

 NAME   PG DESCRIP                      NAME   PG DESCRIP

 ------------------------------- Full-Text News -------------------------------
 CPLAWR  3 Computer Lawyer              DATA    3 Data Comm
 CMPPRO  3 Computer Protocols           DISTEC  3 Display Tech
 CRN     3 Comp Reseller News           DOTCOM  3 dot.com
 CRWEEK  3 Computer Retail Week         EBUSAD  3 e-Business Advisor
 CMPWKY  3 Computer Weekly              ELEBRF  3 Electronic Commerce Briefing
 CMPWS   3 Computer Workstations        ELCMFG  3 Electro Manufacturing
 COMPUT  3 Computers Today              EBNEWS  3 Electronic Buyers' News
 CMPWLD  3 Computerworld                EETMS   3 Electronic Engineering Times
 COMPTG  3 Computing                    EMTECH  3 Electronic Materials
 COMPJP  3 Computing Japan              ELCMSG  3 Electronic Mesg.
 CD2000  3 Countdown 2000               EMEDIA  3 Electronic Media
 M2CIU   3 Corp IT Update
SEE NEXT PAGE FOR ADDITIONAL SOURCES.
```

```
Please ENTER, separated by commas, the NAMES of the files you want to search.
You may select as many files as you want, including files that do not appear
below, but you must enter them all at one time.  To see a description of a
file, ENTER its page (PG) number.
            FILES - PAGE 4 of 10 (NEXT PAGE or PREV PAGE for additional files)

 NAME   PG DESCRIP                      NAME   PG DESCRIP

 -------------------------------- Full-Text News -------------------------------
 ELEWKY  3 Electronics Weekly           INTWLD  5 Internet World
 EXE     3 EXE                          INTRTC  5 Intertec
 FEDTEC  4 Federal Tech Report          ISDNNW  5 ISDN News
 GUIPNW  4 GUI Program News             ISPBUS  5 ISP Business News
 HMOFC   4 Home Office Computing        M2ITC   5 IT Contracts
 IMGUPD  4 Imaging Update               JAVAWD  5 JavaWorld
 INFSEC  4 Info Security                LANPNW  5 LAN Product News
 INFOWK  4 InformationWeek              LANTME  5 LAN Times
 INFWRV  4 Information World Review     LMRNWS  5 LNDMBL Radio News
 INFDLY  4 InfoWorld Daily              MNFMCP  5 Mainframe Computing
 M2IBN   5 Internet Bus New             M2MM    5 Mobile Matters
 INTWK   5 CommunicationsWeek           MOBPHN  5 Mobile Phone Nws
SEE NEXT PAGE FOR ADDITIONAL SOURCES.
```

```
Please ENTER, separated by commas, the NAMES of the files you want to search.
You may select as many files as you want, including files that do not appear
below, but you must enter them all at one time.  To see a description of a
file, ENTER its page (PG) number.
            FILES - PAGE 5 of 10 (NEXT PAGE or PREV PAGE for additional files)

 NAME   PG DESCRIP                      NAME   PG DESCRIP

 ------------------------------- Full-Text News -------------------------------
 MODEMU  5 Modem User News              PCBUSP  5 PC Business Products
 NTWKCP  5 Network Computing            PCWLD   6 PC World
 NETNEW  5 Network News                 PHILPS  6 Phillips Nwslttrs
 NETSOL  5 Network Solutions            PROSFT  6 Productivity Software
 NTWKUP  5 Networks Update              RCR     6 RCR Radio Com. Rpt.
 NWW     5 Network World                RBOCUP  6 RBOC Update
 NWSBYT  5 Newsbytes                    SATNWS  6 Satellite News
 NZTECH  5 NZ Infotech Weekly
SEE NEXT PAGE FOR ADDITIONAL SOURCES.
```

```
Please ENTER, separated by commas, the NAMES of the files you want to search.
You may select as many files as you want, including files that do not appear
below, but you must enter them all at one time.  To see a description of a
file, ENTER its page (PG) number.
            FILES - PAGE 6 of 10 (NEXT PAGE or PREV PAGE for additional files)

 NAME   PG DESCRIP                        NAME   PG DESCRIP

 ------------------------------- Full-Text News -------------------------------
 SATDR   6 Satellite Trader              UNIXUP  6 Unix Update
 SUNWLD  6 SunWorld                      VARBUS  6 VARBusiness
 TECHWB  6 TechWeb News                  WINMAG  6 Windows Magazine
 TCOM    6 tele.com                      M2WCPN  6 Worldwide Comput
 M2TWW   6 Telecomworldwire              WWDB    6 Worldwide Databases
 TIPNWS  6 Telephone IP News             WWTLCM  6 Worldwide Telecom
 TELPNS  6 Tele-Service News             WWVUP   6 Worldwide Videotex Update

SEE NEXT PAGE FOR ADDITIONAL SOURCES.
```

```
Please ENTER, separated by commas, the NAMES of the files you want to search.
You may select as many files as you want, including files that do not appear
below, but you must enter them all at one time.  To see a description of a
file, ENTER its page (PG) number.
            FILES - PAGE 9 of 10 (NEXT PAGE or PREV PAGE for additional files)

 NAME   PG DESCRIP                      NAME   PG DESCRIP

 ------------------ Selected Computer/Communications News ---------------------
 FRESNO  9 Fresno Bee                   INVDLY  9 Investors Daily
 IACCE   9 IAC Computers & Elec         ITIMES  9 The Irish Times
 IACTL   9 IAC Telecommunications       JANDEF  9 Janes's Defence Pub.
 INC     9 INC.                         JOC     9 Journal of Commerce
 INDPNT  9 The Independent              KCSTAR  9 Kansas City Star
 INNEWS  9 The Indianapolis News        KRTBUS  9 Knight Ridder
 INDWK   9 Industry Week                LGECON  9 Law Practice Management
                                        M2COM   9 M2 Communications
                                        MILLFR  9 Miller Freeman
                                        CNGDLY  9 Nat. Jnl. CongressDaily

SEE NEXT PAGE FOR ADDITIONAL SOURCES.
```

```
Please ENTER, separated by commas, the NAMES of the files you want to search.
You may select as many files as you want, including files that do not appear
below, but you must enter them all at one time.  To see a description of a
file, ENTER its page (PG) number.
            FILES - PAGE 10 of 10 (PREV PAGE for additional files)

 NAME   PG DESCRIP                      NAME   PG DESCRIP

 ------------------ Selected Computer/Communications News ---------------------
 NEWSDY 10 Newsday                      SDUT   10 San Diego Union Tribune
 NWEEK  10 Newsweek                     STRAIT 10 Singapore Strait Times
 PBPST  10 Palm Beach Post              USNEWS 10 US News & Wld Rpt
 PHNXGZ 10 Phoeniz Gazette              VILVOC 10 Village Voice
 PRNEWS 10 PR Newswire                  WTIMES 10 Washington Times
```

PATENT APPLICATION FEE DETERMINATION RECORD

Effective October 1, 1996

Application or Docket Number

76/699

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)	SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
FOR	NUMBER FILED	NUMBER EXTRA	RATE	FEE		RATE	FEE
BASIC FEE				385.00	OR		770.00
TOTAL CLAIMS	17 minus 20 =	* —	x$11=		OR	x$22=	
INDEPENDENT CLAIMS	7 minus 3 =	* 4	x40=		OR	x80=	320
MULTIPLE DEPENDENT CLAIM PRESENT			+130=		OR	+260=	
			TOTAL		OR	TOTAL	1090

* If the difference in column 1 is less than zero, enter "0" in column 2

CLAIMS AS AMENDED - PART II

AMENDMENT A		(Column 1)		(Column 2)	(Column 3)	SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
		CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA	RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
	Total	* 17	Minus	** 20	=	x$11=		OR	x$22=	
	Independent	* 7	Minus	*** 7	=	x40=		OR	x80=	
	FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM					+130=		OR	+260=	
						TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B		(Column 1)		(Column 2)	(Column 3)	SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
		CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA	RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
	Total	*	Minus	**	=	x$11=		OR	x$22=	
	Independent	*	Minus	***	=	x40=		OR	x80=	
	FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM					+130=		OR	+260=	
						TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C		(Column 1)		(Column 2)	(Column 3)	SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
		CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA	RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
	Total	*	Minus	**	=	x$11=		OR	x$22=	
	Independent	*	Minus	***	=	x40=		OR	x80=	
	FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM					+130=		OR	+260=	
						TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

FORM PTO-875 (Rev. 10/96)

*U.S. Government Printing Office: 1996 - 413-288/49191

Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE

MULTIPLE DEPENDENT CLAIM FEE CALCULATION SHEET
(FOR USE WITH FORM PTO-875)

SERIAL NO. 08/761,699

FILING DATE 12/6/96

APPLICANT(S)

CLAIMS

	AS FILED		AFTER 1st AMENDMENT		AFTER 2nd AMENDMENT			*		*		*	
	IND.	DEP.	IND.	DEP.	IND.	DEP.		IND.	DEP.	IND.	DEP.	IND.	DEP.
1	/						51						
2		/					52						
3		/					53						
4		/					54						
5	/						55						
6		/					56						
7		/					57						
8	/						58						
9	/						59						
10		/					60						
11		/					61						
12		/					62						
13		/					63						
14	/						64						
15		/					65						
16	/						66						
17	/						67						
18							68						
19							69						
20							70						
21							71						
22							72						
23							73						
24							74						
25							75						
26							76						
27							77						
28							78						
29							79						
30							80						
31							81						
32							82						
33							83						
34							84						
35							85						
36							86						
37							87						
38							88						
39							89						
40							90						
41							91						
42							92						
43							93						
44							94						
45							95						
46							96						
47							97						
48							98						
49							99						
50							100						
TOTAL IND.	7	↲		↲		↲	TOTAL IND.		↲		↲		↲
TOTAL DEP.	10						TOTAL DEP.						
TOTAL CLAIMS	17	▨		▨		▨	TOTAL CLAIMS		▨		▨		

PTO-1360 (3-78)

*MAY BE USED FOR ADDITIONAL CLAIMS OR AMENDMENTS

U.S. DEPARTMENT of COMMERCE
Patent and Trademark Office

Form PTO 1130
(REV 2/94)

PACE DATA ENTRY CODING SHEET

U.S. DEPARTMENT OF COMMERCE
Patent and Trademark Office

1ST EXAMINER GREENE DATE 2/5/97

2ND EXAMINER 500 DATE 5-14-97

APPLICATION NUMBER	TYPE APPL	FILING DATE MONTH DAY YEAR	SPECIAL HANDLING	GROUP ART UNIT	CLASS	SHEETS OF DRAWING
08/761699	1	12 06 96	0	2415	395	7

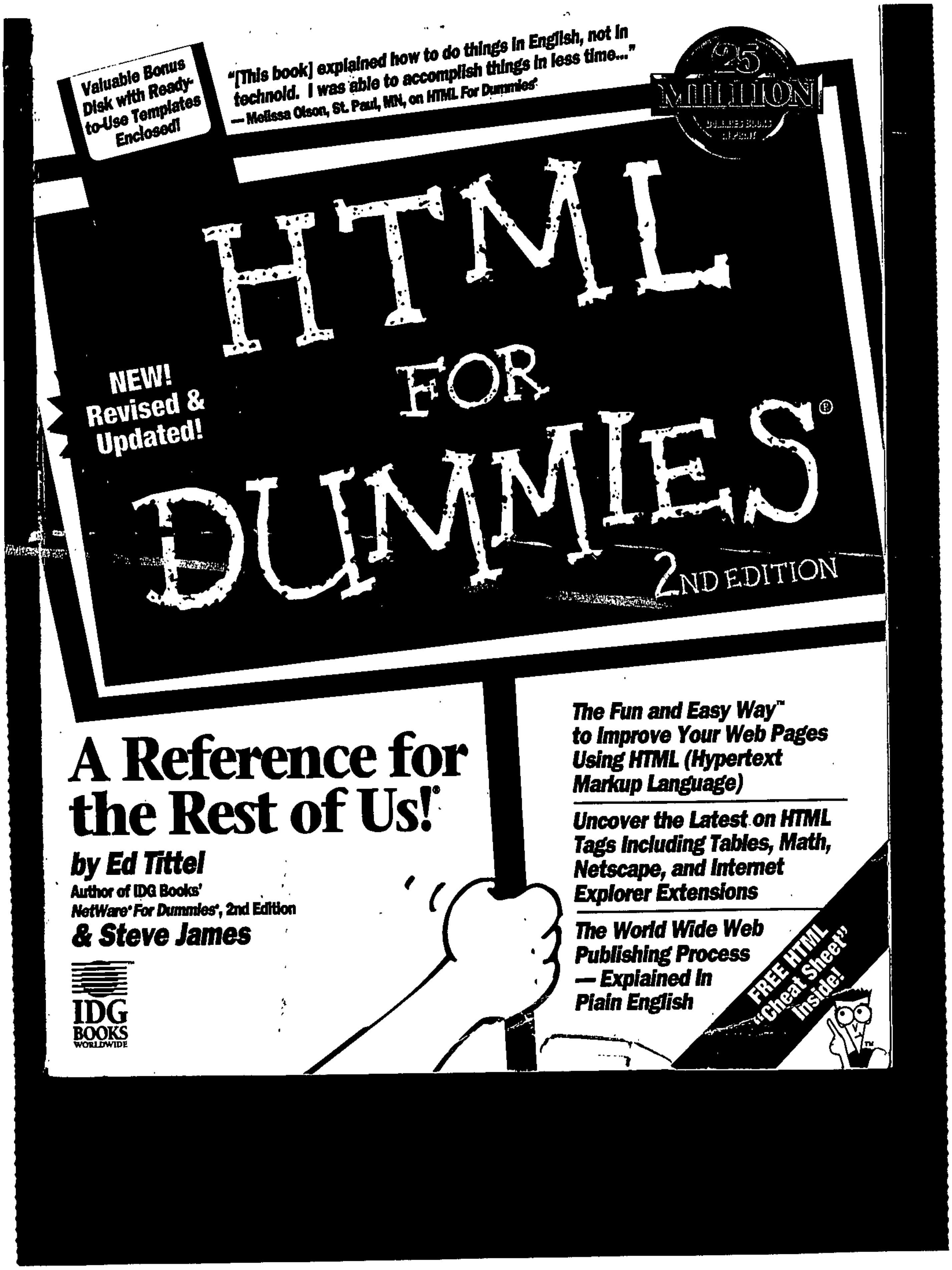
TOTAL CLAIMS	INDEPENDENT CLAIMS	SMALL ENTITY?	FILING FEE	FOREIGN LICENSE	ATTORNEY DOCKET NUMBER
17	7		1220	Y	3382-45418

CONTINUITY DATA

CONT CODE	STATUS CODE	PARENT APPLICATION SERIAL NUMBER	PCT APPLICATION SERIAL NUMBER	PARENT PATENT NUMBER	PARENT FILING DATE MONTH DAY Y
			PCT/ /		
			PCT/ /		
			PCT/ /		X
			PCT/ /		
			PCT/ /		

PCT/FOREIGN APPLICATION DATA

FOREIGN PRIORITY CLAIMED	COUNTRY CODE	PCT/FOREIGN APPLICATION SERIAL NUMBER	FOREIGN FILING DATE MONTH DAY YEAR



Valuable Bonus Disk with Ready-to-Use Templates Enclosed!

"[This book] explained how to do things in English, not in technoid. I was able to accomplish things in less time..."
— Melissa Olson, St. Paul, MN, on *HTML For Dummies*®

25 MILLION

HTML FOR DUMMIES®

2ND EDITION

NEW! Revised & Updated!

A Reference for the Rest of Us!®

by Ed Tittel
Author of IDG Books' *NetWare® For Dummies®*, 2nd Edition
& Steve James

IDG BOOKS WORLDWIDE

The Fun and Easy Way™ to Improve Your Web Pages Using HTML (Hypertext Markup Language)

Uncover the Latest on HTML Tags Including Tables, Math, Netscape, and Internet Explorer Extensions

The World Wide Web Publishing Process — Explained in Plain English

FREE HTML "Cheat Sheet" Inside!

HTML FOR DUMMIES®

2ND EDITION

by Ed Tittel and Steve James

IDG BOOKS WORLDWIDE

IDG Books Worldwide, Inc.
An International Data Group Company

Foster City, CA ♦ Chicago, IL ♦ Indianapolis, IN ♦ Southlake, TX

HTML For Dummies®, 2nd Edition

Published by
IDG Books Worldwide, Inc.
An International Data Group Company
919 E. Hillsdale Blvd.
Suite 400
Foster City, CA 94404
http://www.idgbooks.com (IDG Books Worldwide Web site)
http://www.dummies.com (Dummies Press Web site)

Summer 11, 1996 publication

Library of Congress Catalog Card No.: 96-75002

ISBN: 1-56884-647-9

Printed in the United States of America

10 9 8 7 6 S 4 3

2E/QR/QX/ZX/IN

Distributed in the United States by IDG Books Worldwide, Inc.

Distributed by Macmillan Canada for Canada; by Transworld Publishers Limited in the United Kingdom and Europe; by WoodsLane Pty. Ltd. for Australia; by WoodsLane Enterprises Ltd. for New Zealand; by Longman Singapore Publishers Ltd. for Singapore, Malaysia, Thailand, and Indonesia; by Simron Pty. Ltd. for South Africa; by Toppan Company Ltd. for Japan; by Distribuidora Cuspide for Argentina; by Livraria Cultura for Brazil; by Ediciencia S.A. for Ecuador; by Addison-Wesley Publishing Company for Korea; by Ediciones ZETA S.C.R. Ltda. for Peru; by WS Computer Publishing Company, Inc., for the Philippines; by Unalis Corporation for Taiwan; by Contemporanea de Ediciones for Venezuela. Authorized Sales Agent: Anthony Rudkin Associates for the Middle East and North Africa.

For general information on IDG Books Worldwide's books in the U.S., please call our Consumer Customer Service department at 800-762-2974. For reseller information, including discounts and premium sales, please call our Reseller Customer Service department at 800-434-3422.

For information on where to purchase IDG Books Worldwide's books outside the U.S., please contact our International Sales department at 415-655-3172 or fax 415-655-3295.

For information on foreign language translations, please contact our Foreign & Subsidiary Rights department at 415-655-3021 or fax 415-655-3281.

For sales inquiries and special prices for bulk quantities, please contact our Sales department at 415-655-3200 or write to the address above.

For information on using IDG Books Worldwide's books in the classroom or for ordering examination copies, please contact our Educational Sales department at 800-434-2086 or fax 817-251-8174.

For press review copies, author interviews, or other publicity information, please contact our Public Relations department at 415-655-3000 or fax 415-655-3299.

For authorization to photocopy items for corporate, personal, or educational use, please contact Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, or fax 508-750-4470.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: AUTHOR AND PUBLISHER HAVE USED THEIR BEST EFFORTS IN PREPARING THIS BOOK. IDG BOOKS WORLDWIDE, INC., AND AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS BOOK AND SPECIFICALLY DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THERE ARE NO WARRANTIES WHICH EXTEND BEYOND THE DESCRIPTIONS CONTAINED IN THIS PARAGRAPH. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES REPRESENTATIVES OR WRITTEN SALES MATERIALS. THE ACCURACY AND COMPLETENESS OF THE INFORMATION PROVIDED HEREIN AND THE OPINIONS STATED HEREIN ARE NOT GUARANTEED OR WARRANTED TO PRODUCE ANY PARTICULAR RESULTS, AND THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY INDIVIDUAL. NEITHER IDG BOOKS WORLDWIDE, INC., NOR AUTHOR SHALL BE LIABLE FOR ANY LOSS OF PROFIT OR ANY OTHER COMMERCIAL DAMAGES, INCLUDING BUT NOT LIMITED TO SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR OTHER DAMAGES.

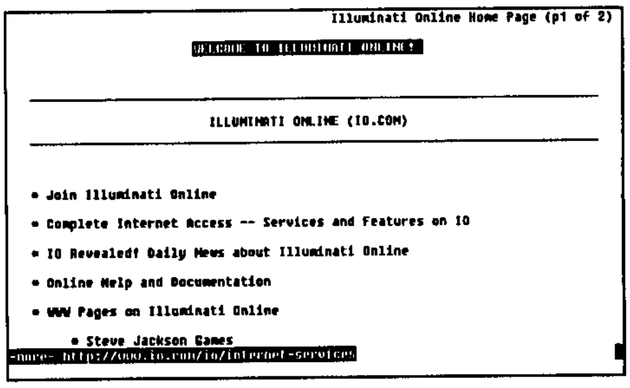
Any one reason would make the Web important and useful; all of them together make it a genuine step forward in the way we use and share information as a part of our daily lives.

Of Browsers and Search Tools

For most end users, their Web access software — called a *browser* or a Web client — is the most important piece of Internet software that they use. Today, you can find many browsers for PCs running Windows, a more limited selection for DOS-only machines, and several options for the Macintosh, UNIX machines, and other platforms. All graphical Web browsers share a common, point-and-click approach to interacting with information. Even character-based browsers, like Lynx, still make it easy to pick and follow links by selecting the appropriate highlighted text. In the next few paragraphs, we give you a cursory look at browsers.

Lynx

Lynx is a text-only Web browser. That is, it cannot display or deliver graphical or multimedia elements (although it can be configured to display graphics using an external file viewer on appropriate systems). Even so, Lynx provides useful Web functionality for users on so-called *dumb terminals* because it supports keyboard navigation and boldface display of hypertext links (which is where we think that the program got its name: Lynx = links; get it?). Figure 1-4 shows a sample Lynx display.



```
                                        Illuminati Online Home Page (p1 of 2)
                        WELCOME TO ILLUMINATI ONLINE!

                        ILLUMINATI ONLINE (IO.COM)

 • Join Illuminati Online
 • Complete Internet Access -- Services and Features on IO
 • IO Revealed! Daily News about Illuminati Online
 • Online Help and Documentation
 • WWW Pages on Illuminati Online
      • Steve Jackson Games
-more- http://www.io.com/io/internet-services
```

Figure 1-4: Lynx uses text displays effectively for Web pages.

Netscape

Netscape, the brainchild of Marc Andreesen, is the Internet's most popular graphical Web browser. It offers clear evidence of its developer's wisdom and experience, and it includes some advanced features not found in other Web browsers. The list of features, advancements, and add-ons to Netscape is a list that boggles the mind — with additions made almost daily. Available both as shareware and in a commercial release, Netscape provides one of the best and most popular Web interfaces that we've encountered anywhere. See Figure 1-5 for a look at Netscape's welcome screen. Visit Netscape at `http://www.netscape.com`.

Internet Explorer

Like a juggernaut, Microsoft's Internet Explorer has taken the Net by storm. Internet Explorer is taking the bite out of Netscape's market share and, if Bill gets his way, will soon be the Web browser of choice. Originally developed for Windows 95, Microsoft has released versions for all Windows platforms and for Macintosh. You can find the latest release of Internet Explorer at the Microsoft Web site: `http://www.microsoft.com`.

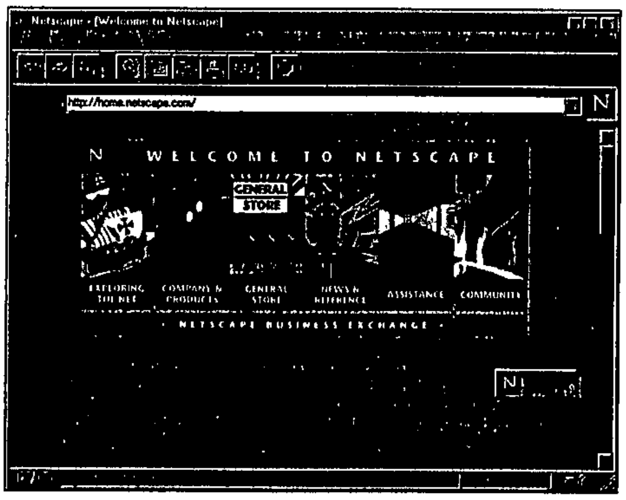
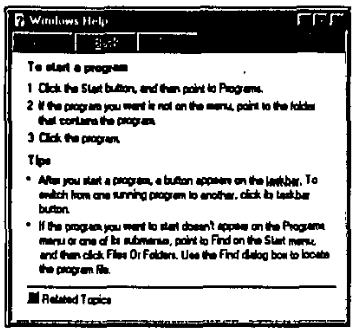


Figure 1-5: Netscape is used by over 70 percent of Web surfers.



To start a program

1 Click the Start button, and then point to Programs.
2 If the program you want is not on the menu, point to the folder that contains the program.
3 Click the program.

Tips

- After you start a program, a button appears on the taskbar. To switch from one running program to another, click its taskbar button.
- If the program you want to start doesn't appear on the Programs menu or one of its submenus, point to Find on the Start menu, and then click Files Or Folders. Use the Find dialog box to locate the program file.

Related Topics

Figure 2-2: The Microsoft Windows Help engine is a well-known example of hypertext.

and linkage, and integrated multimedia help to realize the notion of hypertext and how it operates.

The difference between the preceding examples and the Web is the kinds of links that they support: Only the Web offers the capability of jumping across the Internet and following links to other Web documents and servers. Enabling this feature is what gives HTML its unique value and, of course, what's made the Web possible.

Beyond Text, There's Multimedia

If you include nontext files (such as sound, graphics, and video) in Web pages, you must employ a certain amount of alchemy. Shipping Web information in a format called *MIME* (Multipurpose Internet Mail Extensions) makes it possible for a Web server to deliver multiple forms of data to your browser in a single transfer. Actually, MIME is a technique designed to bundle attachments within individual e-mail files. The following paragraphs tell a little more about how this works.

After a MIME file with attachments shows up at your workstation, additional processing begins immediately. The text portion of the message file arrives first. It contains the text-only HTML page description and lets the browser get right to work building and displaying the text portions of the page. Because the text arrives first, you may see placeholders or icons for graphics when you first see the Web page. Then the graphics or other forms of data replace these placeholders after their related attachments arrive.

While the user views the Web page, the browser receives attachments in the background. As they come in, the browser identifies these attachments by a file type or by description information in the attachment tag (as specified by the MIME format). After the browser identifies a file, it can handle the file's playback or display. Table 2-1 shows a list of common file types used on the Web, including expansions for the inevitable acronyms that such files often entail.

Table 2-1 Common Sound, Graphics, and Motion-Video Formats on the Web

Extension	*Format*	*Explanation*
Sound Formats		
RA	RealAudio	Used with RealAudio Web Server and RealAudio Player add-on for browsers.
SBI	Sound Blaster Instrument	Used for a single instrument with Sound Blaster cards (multiinstrument: IBK).
SND, AU	8 kHz mulaw	Voice-grade sound format used on workstations (such as Sun, NeXT, HP9000, and so on).
WAV	Microsoft Waveform	Sound format used in Windows for event notification.
Still-Video (Graphics) Formats		
GIF	Graphics Interchange Format	Compressed graphics format commonly used on CompuServe, easy to render multiplatform. Can be interleaved or not, depending on how image is created.
JPEG, JPG	Joint Photographic Experts Group	Highly compressed format for still images, widely used for multi-platform graphics.
PDF	Portable Document Format	Adobe's format for multiplatform document access through its Acrobat software.
PS	PostScript	Adobe's type description language, used to deliver complex documents over the Internet.
XBM	X-Window Bitmap	Image bitmap used by X-Windows, primarily on UNIX workstations.
Motion-Video Formats		
AVI	Audio Video Interleaved	Microsoft's Video for Windows standard format; found on many CD-ROMs.
DVI	Digital Video Interactive	Another motion-video format, also found on CD-ROMs.

[redacted]

Extension	*Format*	*Explanation*
FLI	Flick	Autodesk Animator motion-video format.
MOV	QuickTime	Apple's motion video and audio format; originated on the Macintosh, but also available for Windows.
MPEG, MPG	Motion Picture Experts Group	Full-motion video standard using frame format similar to JPEG with variable compression capabilities.

Several Web sites contain large amounts of information on file formats and programs.

- **Common Internet File Formats:** Contains an annotated list of audio, graphic, and multimedia file formats with links to applications that use them.

  [redacted]

  [redacted]

- **GRAPHICS:** Maintained by Martin Reddy at the University of Edinburgh, this site contains "everything you ever wanted to know" about graphics and links to additional resources.

  [redacted]

- **Multimedia File Formats on the Internet:** A Beginner's Guide for PC Users, by Allison Zhang.

  [redacted]

Hyperhelpers: useful "helper" applications

When referenced by a Web page, nontext data shows up as attachments to the HTML file. Sometimes the browser itself handles playback or display, as often is the case for simple, two-dimensional graphics (including .GIF and .JPEG files). Even so, these may be handled by other applications (especially when using character-mode Web browsers like Lynx).

But when other kinds of files show up and need special handling capabilities (beyond the scope of most browsers), the browser hands off the files to other applications for playback or display. These *helper* applications have the built-in smarts to handle the formats and processing necessary to deliver the contents of specialized files on demand. The process normally works something like this:

- The browser builds a page display that includes an active region (underlined or outlined in some way) to indicate the attachment of a sound, video, or animation playback.
- If the user selects the active region (the link), the browser calls on another application to handle playback or display.
- The other application takes over and plays back or displays the file.
- After the helper completes the display or playback, the browser reasserts control, and the user can continue on (or select the active region again, and get another playback or display).

A standard part of browser configuration supplies the names and locations of helper programs to assist the browser when such data arrives. If the browser can't find a helper application, it simply doesn't respond to an attempt to display or play back the requested information.

For example, WPlany and Wham are two common shareware sound player applications for PCs running Windows. As part of their configuration or setup, most browsers supply a method for linking particular file types (like the .SBI and .WAV file extensions common on the PC) with a helper application.

By establishing an association between the sound playback application (WPlany or Wham) and the related audio file extensions (.SBI and .WAV), the browser automatically launches the application you designate when it encounters files with those extensions. This causes the sounds to play (which, we assume, is a good thing!).

Some Web browser add-ins take the form of a plug-in. A plug-in is simply a helper application which installs into a Web browser rather than just being called by the browser. Plug-ins often use the display window of a browser, while helper applications usually have a separate external display window of their own.

For comprehensive listings of PC, Mac, and UNIX helper applications and plug-ins, take a look at these URLs:

[illegible]

Some useful helper applications for Windows include

- **For still graphics:** LView is a good, small graphics viewer that can handle .GIF, .PCX, and .JPEG files. It also supports interesting image editing capabilities. (See Chapter 9 for more details.)

- **For video:** You'll want to use QuickTime for Windows (for QuickTime movies) or MPEGplay for .MPEG video files.
- **For PostScript viewing:** Ghostview for Windows works with a companion program called Ghostscript that allows users to view or print PostScript files from any source, including the Web. Because so many documents on the Internet have the PostScript format, we find these to be useful programs.

All in all, a good set of helper applications can make your browser even more effective at bringing the wonders of the Web to your desktop. With the right help, your browser can play back or render just about anything you'll run into!

The Value of Visuals

Without a doubt, graphics add impact and interest to Web pages, but that extra punch doesn't come without a price. You can easily get carried away by the appeal of pictures and overdo their use on your Web page. (Overdoing applies as much to the small images that you use as buttons and visual on-screen controls as it does to large images that help to dress up the Web page.)

Therefore, when you use graphics, remember these two important things:

- Not everybody who reads your page can see the graphics. Readers may not see the graphics because they use a character-mode browser (that can't display them) or because they switch off their graphics display (a common option on most Web browsers to conserve *bandwidth,* or their ability to move data, and improve response time).
- Graphics files — even compressed ones — can sometimes be quite large, often ten or more times greater in size than the HTML file that they're attached to. Moving graphics across the network takes time and consumes bandwidth. Also, it penalizes users with slower modems far more than those attached directly to the Internet.

Sometimes graphics are essential — for example, when you are using a diagram or illustration to explain your material. In other situations, impact is important — such as on a home page, where you'll be making a first impression. Under these circumstances, using graphics is perfectly appropriate, but be sensitive to the different capabilities and bandwidths of your readers.

TIP

Following are rules of thumb for using graphics effectively in your Web pages. (As you examine the work of others, see what happens to your attitude when you find that they violated these rules.)

- Keep your graphics small and uncomplicated whenever possible. This reduces file sizes and keeps transfer times down.
- Keep file sizes smaller by using compressed formats (like .GIF and .JPEG) when you can.

- Create a small version (called a *thumbnail*) of your graphic to include in your Web page. If you must use larger, more complex graphics, link the created thumbnail on your Web page to the full-size version of the graphic elsewhere. This linking spares casual readers the impact of downloading the larger version every time they access the page (and keeps Internet usage under better control, making you a better *netizen!*).
- Keep the number of graphic elements on a page to a minimum. Practically speaking, this means at most two or three graphic items per page, where one or two items are compact, iconlike navigation controls and another is a content-specific graphic. Here again, the idea is to limit page complexity and to speed up transfer times.

Sometimes, the temptation to violate these rules of thumb is nearly overwhelming. If you must break the rules, be sure to run your results past some disinterested third parties. (You'll learn more about testing techniques in Part V of this book.) Watch them read your pages if you can. Listen carefully to their feedback to see whether you've merely bent these rules or broken them to smithereens!

Also, remembering that not everybody who accesses the Web can see your graphics should help to keep you humble. For readers who don't have access to graphical browsers, try to think of ways to enhance their reading experience even without graphics.

Mavens of multimedia

The rules that go for graphics go *double* for other forms of multimedia. If graphics files are large when compared to HTML text files, then sound and video files are HUGE. They are time dependent; therefore, the longer they play, the bigger they are, and the longer the browser takes to download them to your computer. Although they're appealing and definitely increase the interest level for some topics, sound and video files are not germane to most topics on the Web. Use them sparingly or not at all, unless your Web site is an Internet radio show or movie theater.

With the advent of the smaller, faster display and interactivity tools for the Web (such as Java, VRML, and Shockwave), the Web will undoubtedly use more and more multimedia applications. Java *applets* work with Netscape, Mosaic, and other browsers to allow quick display of animated graphics and other special effects. VRML (Virtual Reality Modeling Language) is similar to HTML but provides 3-D viewing capabilities and more. Shockwave is the name of Macromedia's plug-in to Netscape 2.0 that allows Director movies to play inside Web pages. The use of these tools goes far beyond the scope of this book, but you can look forward to using them in your second generation Web page. If you're interested in these tools, you can get more information at the following URLs:

Java — Sun Microsystems

http://www.javasoft.com or http://java.sun.com

VRML

http://www.vrml.org/

Shockwave — Macromedia

http://www.macromedia.com

Once again, the trick is to make large files available through links instead of including them on pages that everyone tries to download. Labeling such active regions with the file's size is also a good idea so that people know what they're in for if they choose to download. (For example: `Warning! This points to a 40K sound byte of a barking seal.`)

Bringing It All Together with the Web

Let's get down from multimedia hyperspace and back to the cyberspace world of your future Web page. Now that you understand the basic concepts behind the Web and HTML and have met (briefly) some tools of the trade, you need to read the following paragraph carefully. It embodies the essence of your Web pages.

The three most important factors in building good Web pages are content, content, and content. (Get the idea?) If the content is well-organized, engaging, and contains links to interesting places, your Web site can be a potent tool for education and communication. If your Web site is all flash, sharing it can be an exercise in sheer frustration (and humiliation for the WebMaster . . . that's you!). Therefore, if you put your energy into providing high-quality content and link your readers to other high-quality, content-filled pages, your Web site will be a howling success. If you don't, your site will be the electronic equivalent of a ghost town!

In the next chapter, you'll discover what's involved in using — and building — documents for the Web. Just fly into our parlor for a look at what's in (and on) a Web page.

Chapter 4

What's a Markup Language?

In This Chapter

- ▶ Defining HTML markup conventions
- ▶ Understanding HTML's roots
- ▶ Introducing the basic HTML control characters, elements, attributes, and entities

In this chapter, you'll finally see what HTML looks like. You'll begin to appreciate what's involved in a markup language and start to understand how to use HTML to create your own Web pages. Because this chapter is an overview, you won't be able to run right out and start building pages after you read it, but you should have a pretty good idea of the pieces and parts that make HTML do its thing.

A Markup Language Is Not a Form of Graffiti

HTML represents a way to take ordinary text and turn it into hypertext, just by adding special elements that instruct Web browsers how to display its contents. These special elements are called markup tags.

The purpose of a markup language is to give either machines or humans clues about the structure, content, and behavior of a document. Markup comes in two types: descriptive and procedural. HTML is a descriptive markup language. A descriptive markup language describes the structure and behavior of a document. This focus allows an author to concentrate more on content and structure, and less on formatting and presentation. Here's an example:

```
<H1>[illegible] - display level one heading [illegible]
<OL> - begin ordered (numbered) list [illegible]
<LI> take [illegible]
```

```
[illegible]
[illegible]
[illegible]
```

The tags H1, OL, and LI describe an object and its components. It's up to a browser to render these properly on your screen. In fact, from browser to browser, each of these objects could be rendered differently. But the important thing is that each browser has a meaningful way to display a first-level header (H1) and a numbered list (OL and LI). From a portability standpoint, this is a good thing.

The other kind of markup language, procedural, describes formatting rather than structure. An example is *troff*, the dinosaur UNIX màrkup language, which looks like this:

```
[illegible] Bold Table of Contents
```

These *troff* tags tell the output device to render an object as "centered, 12 point Helvetica bold":

Table of Contents

As you can imagine, this type of markup is not very portable and is difficult to maintain.

HTML tags not only govern how a browser displays the contents of an HTML document, but also control how the browser can draw in graphics, video, sound, and so on to create a multimedia experience.

In the same vein, still other HTML tags instruct browsers how to handle and display hypertext links, either within the same HTML file or to other documents or Web-accessible services. The key to building attractive, readable Web pages is knowing how to use HTML markup to highlight and organize your content.

A syntax is not a levy on cigarettes!

When it comes to any kind of formal, computer-readable language, you invariably find a set of rules governing its terms and their order of placement. This set of rules is called the *syntax* of the language. In HTML the syntax describes how a Web browser can recognize and interpret the instructions contained in the markup tags.

What makes HTML particularly interesting is that it's all pure text — in fact, HTML can work completely within the confines of the ASCII 7-bit character set (ISO 646), which contains only 128 distinct viewable characters. Nevertheless, HTML can handle display of so-called higher-order ASCII characters, which normally require eight bits to represent directly and are sometimes called the 8-bit ASCII or Latin-1 character set (ISO 8859/1). Quite a bit of work is under way to make HTML able to handle the Unicode character set, which supports up to 65,535 different characters (of which over 34,000 are defined). The Unicode set includes non-Roman alphabets such as Chinese (Han) ideograms, Hebrew characters, and lots of other interesting stuff.

This diversity lets HTML display things like accents, umlauts, and other diacritical marks often associated with non-English languages (or loan words, like résumé, from those languages), by including instructions on what characters to represent as a part of the markup. In other words, HTML can provide the related instructions to a browser even if you can't always see things in the same format when you're writing the HTML. For example, <OL> and <LI> don't look like a numbered list inside your text file, but they do when the browser interprets them.

Elements of HTML syntax

The special control characters that separate HTML markup from ordinary text are the left and right angle brackets: < (left bracket) and > (right bracket). These characters indicate that the browser should pay special attention to what they enclose. And here's your big chance to learn another buzzword: *parser.* Inside the browser software, a parser does the work of reading and constructing the display information. A parser reads information in the HTML file and decides which elements are markup and which ones aren't, permitting the browser to take appropriate action.

In HTML, left and right angle brackets can enclose all kinds of special instructions called *tags*. The next six chapters of this book are devoted to introducing, explaining, and demonstrating HTML tags.

In the meantime, here's a general introduction to the way HTML tags look and behave: A tag takes a generic form that looks something like this:

```
<TAG NAME ATTRIBUTE="VALUE" ...> ... text ... </TAG NAME>
```

(By the way, text within the tags is case insensitive, but for readability, we use all caps to make HTML tags stand apart from other text.)

Let's look at the pieces of this generic form:

- <TAG-NAME>: All HTML tags have names. For example, H1 is a level-one header; OL is an ordered list. Tags are surrounded by angle brackets that mark their contents for special attention from the parser software.
- {ATTRIBUTE{="VALUE"}...}: Some HTML tags require or permit certain named attributes to be associated with them. Here, the notation using curly braces — for example, {ATTRIBUTE} — indicates that attributes may be present for some tags, but not for others. In the same vein, some attributes require that values be associated to them, as when supplying the name of an HTML document for a hypertext link; other attributes may not require associated values at all, which is why {="VALUE"} is also in curly braces.

  For example, in an <IMG> tag used to point to a graphics file, a required attribute is SRC (source), to provide a pointer to the file where the graphic resides, as in <IMG SRC="../gifs/redball.gif." This syntax proves yet again how important it is to stay close to your source! Finally, the ellipsis (...) indicates that some HTML tags may even include multiple attributes, each of which may take a value or not, depending on the circumstances.
- Text: This is the content that's modified by a tag. For example, if the tag were a document title, the HTML string

  ```
  <TITLE>HTML For Dummies Home Page</TITLE>
  ```

  would display the words "HTML For Dummies Home Page" in the title bar at the top of a graphical browser's window. Enclosing this text within <TITLE> and </TITLE> tags marks it as the title for the document.
- {</TAG-NAME>}: The closing tag name is denoted by the left angle bracket (<), followed by a forward slash (/), the tag name, and finally a closing right angle bracket (>). The curly braces indicate that this element does not always occur. However, over 70 percent of the HTML tags require a closing tag as well as an opening tag, so the omission of a closing tag should be considered the exception, rather than the rule. Don't worry; most, if not all, browsers will simply ignore closing tags if they're not necessary.

The majority of HTML tags don't require the assignment of attributes, so don't be too overwhelmed by their full-blown formal syntax. By and large, most tags resemble the <TAG-NAME>Text</TAG-NAME> layout like <TITLE>HTML For Dummies Home Page</TITLE>.

The ampersand (&) is another special HTML control character. It's used to denote a special character for HTML content that might not belong to the 7-bit ASCII character set (like an accent grave or an umlaut), or that might otherwise be interpreted as a markup character (like a left or right angle bracket). Such tagged items are called character entities, and can be expressed in a number of ways. For example, the string &egrave; produces a lowercase E with a grave accent mark (è), while the string &lt; produces a left angle bracket (<).

Table 5-2 *(continued)*

Category/ Tags	*Tag Names*	*Category Description/ Brief Explanation*
<ISINDEX>	Isindex	Indicates that document supports CGI script for searches
<LINK>	Linktype	Sets relationship between current document and other documents
<NEXTID>	Next document	Indicates the "next" document that follows current, to permit HTML documents to be chained together
<META>	Structure	Describes aspects of the page's info structure, contents, or relationships to other documents
Document Headings		Supply document title and heading levels, provide important organization and layout elements
<TITLE>...</TITLE>	Title	Supplies title that labels entire document
<H1>...</H1>	Level 1 head	First-level heading
<H2>...</H2>	Level 2 head	Second-level heading
<H3>...</H3>	Level 3 head	Third-level heading
<H4>...</H4>	Level 4 head	Fourth-level heading
<H5>...</H5>	Level 5 head	Fifth-level heading
<H6>...</H6>	Level 6 head	Sixth-level heading
Links		Create links to anchor or another document, or create anchor point for another link
<A>...</A>	Anchor or link	Provides fundamental hypertext link capabilities
Layout Elements		Control document appearance, add elements
<ADDRESS>...</ADDRESS>		Author contact information for document
<BLOCKQUOTE>...</BLOCKQUOTE>		Use to set off long quotes or citations
<BR>	Line break	Forces a line break into on-screen text flow
<PRE>...</PRE>	Preformatted text	Preserves spacing and layout of original text in monospaced font
<HR>	Horizontal rule	Draws a horizontal line across the page

Category/ Tags	*Tag Names*	*Category Description/ Brief Explanation*
Graphics		References to inline images for documents
<IMG>	Image	Inserts a referenced image into a document with alternate text, clickable map, and placement controls
Forms		Forms-related markup tags
<FORM>...</FORM>	Form block	Marks beginning and end of form block
<INPUT>	Input widget	Defines type and appearance for input widgets
<TEXTAREA>... </TEXTAREA>	Text area	Multiline text entry widget
<SELECT>... </SELECT>	Input pick list	Creates a menu or scrolling list of input items
<OPTION> [...</OPTION>]	Selectable item	Assigns a value or default to an input item
Paragraphs		Break up running text into readable chunks
<P>	Paragraph	Breaks up text into spaced regions
Lists		Provide methods to lay out item or element sequences in document content
<DIR>...</DIR>	Directory list	Marks unbulleted list of short elements (less than 20 characters in length)
<LI>	List item	Marks a member item within a list of any type
<OL>...</OL>	Ordered list	Marks numbered list of elements
<UL>...</UL>	Unordered list	Marks bulleted list of elements
<MENU>...</MENU>	Menu list	Marks a pickable list of elements
<DL>...</DL>	Glossary list	Marks a special format for terms and their definitions
<DT>	Definition term	Marks the term being defined in a glossary list
<DD>	Definition datum	Marks the definition for a term in a glossary list
Text Controls		Character formatting tags
<B>...</B>	Boldface	Produces bolded text
<CITE>...</CITE>	Short citation	Marks distinctive text for citations
<CODE>...</CODE>	Code font	Used for code samples

(continued)

Table 5-2 *(continued)*

Category/ Tags	*Tag Names*	*Category Description/ Brief Explanation*
Text Controls *(cont'd)*		Character formatting tags
<DFN>...</DFN>	Defined term	Used to emphasize a term about to be defined in the following text
<EM>...</EM>	Emphasis	Adds emphasis to enclosed text
<I>...</I>	Italic	Produces italicized text
<KBD>...</KBD>	Keyboard text	Marks text to be typed at keyboard
<SAMP>...</SAMP>	Sample text	Marks sample in-line text
<STRONG>...</STRONG>	Strong emphasis	Provides maximum emphasis to enclosed text
<TT>...</TT>	Typewriter text	Produces a typewriter font
<VAR>...</VAR>	Variable	Marks variable or substitution for some other value

Now, let's review the HTML categories that we just introduced, before providing detailed syntax for each tag:

- ✓ **Comments:** Comments give HTML authors a way to annotate their documents, and browsers do not ordinarily display them. Any assumptions, special conditions, or nonstandard elements should be enclosed in comments to help other readers understand what the document is trying to accomplish and to assist with the testing process.
- ✓ **Document Structure:** Numerous tags help to provide structure for HTML documents. They provide an overall HTML label and break up documents into head and body sections. They also provide markup to establish links to other documents and to indicate support for electronic indexing capabilities. While this markup produces little in the way of visible display, it is important to the construction of well-designed Web pages.
- ✓ **Document Headings:** Headings provide structure for a document's content, starting with its title, all the way down to sixth-level headings. They provide meaningful clues for document navigation, and when used in conjunction with a hypertext table of contents, they can permit readers to quickly jump to other sections.
- ✓ **Links:** Links provide the controls to anchor points within a document or to link one document to another. They are the foundation for the Web's hypertext capabilities.

- **Layout Elements:** Layout elements introduce specific items within the text of a document, including line breaks, lengthy quotes, and horizontal rules, to divide up distinct text areas and preformatted text to capture spacing and layout exactly from the source, based on a monospaced font (which makes preformatted text kind of ugly). They also include a format for building author information on a page, which is an element that we recommend for all good Web pages.
- **Graphics:** Graphics enter an HTML file through the `<IMG>` command, which we've already covered in some detail. Suffice it to say that `<IMG>` points to the graphics source, provides a text alternative for nongraphical browsing, and indicates whether a graphical element is a clickable map.
- **Forms:** Forms provide the essential mechanism for soliciting readers, feedback and input on the Web. Forms tags cover how forms are set up, provide a variety of graphical and text widgets for soliciting input, and supply methods to allow readers to select options from various types of pick lists.
- **Paragraphs:** The paragraph is the fundamental unit of text for HTML documents as well as ordinary text. The `<P>` tag allows authors to break their content up into easily digestible chunks.
- **Lists:** HTML includes numerous styles for building lists, ranging from numbered to bulleted lists, glossary entries complete with definitions, and selectable menu entries. All styles provide useful tools to improve readability by organizing lists of items or elements.
- **Text Controls:** HTML also offers numerous inline controls for adding emphasis or special appearance to text. It provides tools for describing user input and for including samples of computer code, computer output, variables, and sample text. You can use text controls to represent different kinds of online text for building materials for online use.

From managing document structure to controlling the look and feel of text on a page, HTML includes tags to make these things happen. In the next section, we examine the nitty-gritty details of all the various HTML 2.0 tags.

HTML Tags

The remainder of this chapter is devoted to an alphabetical listing of a broad range of HTML tags taken from the HTML 2.0 DTD.

Because so many browser-builders are adding extensions to HTML for their own use, and because standards beyond HTML 2.0 have introduced significant changes and enhancements to HTML, you should consider this list neither exhaustive nor complete. We do hope it is informative and useful, however, and we encourage you to skim it over, just to see all the possibilities that HTML offers. For a look at what's beyond HTML 2.0, please check out the disk included with this book (explained in Chapter 12).

The rundown on attributes

In HTML, attributes typically take one of two forms within a tag:

- ATTRIBUTE="value": Where value is typically enclosed in quotes (" ") and may be one of the following kinds of elements:

URL	a Uniform Resource Locator
name	a user-supplied name, probably for an input field
number	a user-supplied numeric value
text	user-supplied text
server	server-dependent name (such as page name defaults)
(X|Y|Z)	one member of a set of fixed values

- ATTRIBUTE: Where the name itself provides information about how the tag should behave (for example, ISMAP in <IMG> indicates that the graphic is a clickable map).

For a discussion of attributes for individual tags, look in the "Attributes" section under the tag name. For each one, we provide a definition. We also indicate choices for predefined sets of values or provide an example for open-ended value assignments.

Tag information layout

Before we provide our alphabetical list of tags, you need to understand what information we're presenting and how it's presented. Using the familiar image tag (<IMG>), we show you what a typical listing looks like:

<IMG>

Definition: Supplies image source, placement, and behavior information. Used to place in-line graphics on a page.

Attributes:

SRC="URL"

URL is a standard Uniform Resource Locator and specifies the location for image file, which is usually .GIF or .JPEG format.

ALT="text"

Supplies an alternate string of text to display (and possibly make clickable) if the browser has no graphics capability or if graphics are turned off.

ALIGN=("TOP"|"MIDDLE"|"BOTTOM") and [WIDTH="number"] andr [HEIGHT="number"]

> Standard use calls for ALIGN to be set to one of the following values: TOP, MIDDLE, or BOTTOM to define placement.
>
> Optional values also permit more precise placement using a pixel-level height and width specification.

ISMAP

> Indicates by its presence that the image (or its text replacement) should be a clickable map. This often invokes special map-handling software through the CGI interface on the Web server handling the request.

Context:

<IMG> is legal within the following markup tags

> <ADDRESS>, <B>, <CITE>, <CODE>, <DD>, <DT>, <EM>, <H*>, <I>, <KBD>, <LI>, <PRE>, <SAMP>, <STRONG>, <TT>, <VAR>
>
> ***Note:*** When referring to heading tags <H1> through <H6> we abbreviate the whole series as <H*> as we did above.

Suggested style/usage: Keep images small and use them judiciously; graphics should add impact and interest to pages without adding too much bulk (or wait time).

Examples:

```
<IMG SRC="images/redball.gif" ALIGN="TOP" WIDTH="50" HEIGHT="50"
ALT="Menu Items">
<IMG SRC="http://www.noplace.com/show-me/pictures/fun.gif"
ALIGN="TOP" ISMAP ALT="Fun places to visit">
```

Tag layout commentary

Notice the use of our HTML syntax notation in the "Attributes" section; you see the syntax here most often. Because <IMG> is a stand-alone tag (that is, there's no closing </IMG>), we don't show a pair of tags here, but we do show tags in pairs whenever appropriate.

The last item for discussion is the "Context" section. In this section, you see where it's legal to put <IMG> tags inside other markup, between <PRE> . . . </PRE> tags, for example. Just because you can use this tag in such a way doesn't mean that you have to do so; as always, use markup judiciously to add impact or value to information. Complex compositions seldom delight anyone other than their makers, so try to keep things simple whenever you can.

Examples:

```
<IMG SRC="images/redball.gif" ALIGN="TOP" ALT="Menu Items">
<IMG SRC="http://www.noplace.com/show-me/pictures/fun.gif"
ALIGN="TOP" ISMAP ALT="Fun places to visit">
```

<INPUT> Input object

Definition: <INPUT> defines an input object within an HTML form; these objects come in several different types, and also include several different ways to name and specify the data they contain.

Attributes:

TYPE =
("TEXT" | "PASSWORD" | "CHECKBOX" | "HIDDEN" | "RADIO" | "SUBMIT" | "RESET")

> Defines the type of input object being described. TEXT, CHECKBOX, and RADIO define how data entry areas appear on-screen; use PASSWORD to prompt for a password; HIDDEN allows the form to pass data to the Web server that users can't see; SUBMIT and RESET provide methods to ship the information on a form to the server, or to clear the data from the form.

NAME="text"

> The name of the input item, as passed to the CGI script for the form as part of a name, value pair (this is how the script identifies values with their corresponding form fields).

VALUE="text"

> The value for the input item, as passed to the CGI script for the form as part of a name, value pair.

SIZE="number"

> The size of a TEXT type input item, as measured by the number of characters it contains.

MAXLENGTH="number"

> The maximum number of characters allowed in a TEXT type input item.

CHECKED

> For checkboxes or radio buttons, inclusion of this attribute indicates that the box was checked or the button selected, usually as a default.

Chapter 7

Building Basic HTML Documents

In This Chapter

▶ Putting it together for the first time

▶ Making templates work for you

▶ Starting page layout at the top

▶ Writing titles and headings with a purpose

▶ Building better bodies for your pages

▶ Building strong paragraphs

▶ Listing with the proper structures

▶ Linking to your Web and beyond

Building your first Web page is exciting if you keep this thought firmly in mind: You can change anything at any time. Good Web pages are always evolving. Nothing is cast in concrete — change is just a keystroke away.

Now that the pressure is off, you can start building your own simple, but complete, home page. Think of it as a prototype for future pages. Later you can add all sorts of bells and whistles to change your home page into any kind of page you want — be it for a business, an academic institution, or a government agency.

The layout, or the way the page looks to the user, creates the first impression of your whole Web site. If that first impression isn't pleasing, the first time may also be the last time that the user visits your page. Not to worry, though: Your home page can be pleasing to the eye if you follow the *KISS* (Keep It Simple, Stupid) approach.

The Web itself is a confusing concept to many users. Everything that you do to keep your page *intuitively obvious* makes your viewers happy and keeps them coming back for more.

Chapter 3 presented the basic concepts of a good Web page. It emphasizes the form and content over the HTML controls and presents the elements of page layout and information flow. You might want to review Chapter 3 before continuing here.

REMEMBER

The basics are content, layout, first impression, and KISS. Okay, now get on with it.

The Template's the Thing!

Well-constructed Web pages contain the following four sections: Title, Heading, Body, and Footer.

If you look at a number of Web pages, you'll undoubtedly see that most contain these sections in one form or another. You may also notice, with some amount of frustration, that the pages that don't contain all these elements either aren't pleasing to your eye or aren't intuitive in their presentation. Plus, you can't easily find what you're looking for. We're not going to let that happen to your pages because you are going to use the following basic template for each HTML document you produce:

```
<HTML>
<HEAD><TITLE>Your Title Here</TITLE>
</HEAD>
<BODY>
<P>
Your headings and body text and graphics go here.
<P>
<ADDRESS>
Your Name<BR>
Phone number<BR>
Standard Mail Address<BR>
E-MAIL Address
</ADDRESS>
<P>
Copyright &copy;1995, Your Name <BR>
Revised - Revision date <BR>
URL <A HREF="http://Your.page's.URL/here">
http://Your.page's.URL.here</A>
</BODY>
</HTML>
```

Getting started on the correct path is really that simple. This template actually works. Figure 7-1 shows what it looks like when viewed with Netscape.

Figure 7-1: The basic Web page template viewed with Netscape.

TIP

- Use your browser to open your Web page HTML document file from your local hard disk.
- If you're using Netscape, remember to set the memory and disk caches to zero so that Netscape reloads each new version of your file from the disk, instead of loading the one in its cache. Other browsers cache pages, too, so make sure that you're reading what you just edited — not some older version!

As you can see, your home page is currently plain and simple. That's not going to have folks flocking to see it, is it? That's because you need to add your own wonderful text and graphics. Even though a growing number of Internet surfers use GUI browsers, please follow our advice (from Chapter 2) and put your energy into providing high-quality content and important links. Don't worry; in the next chapter, you'll add some graphics, too.

Page Layout: Top to Bottom

Now that you have a basic template, you can start changing it. To begin the fun, your first home page shouldn't occupy more than a single screen; this limit makes the page much easier to edit and test. You can get more than enough information on a single screen and help your audience avoid unnecessary scrolling.

A *single screen* seems like an easy concept, but is it? A single screen is the amount of information that a browser can display on the monitor without scrolling. This amount varies depending upon the readers' browsers and their monitor resolutions. You may not want to design for the lowest common denominator for both elements, but please understand the following: If you assume that the user can see your page the same way that you see it in *your* browser, you're making a bad assumption. No easy answer to this problem exists, but testing your pages on a relatively low-resolution monitor with several different browsers can help you see your pages through your reader's eyes. Getting this view is well worth taking the extra time!

Logos

Logos are special-use graphics. They vary from icon size to much larger, sometimes too large. Remember KISS? Complex logos that take too long to load are nugatory on any Web page.

Use logos to identify your business or institution in a pleasing, eye-catching manner. Don't use them to overpower the page or to irritate the users. A moderately sized logo at the top of the home page is generally acceptable. Using icon-sized logos in the footer of each Web page is equally acceptable. Remember that text-only browsers and GUI browsers with image loading turned off (for faster page loading) won't display your fantastic logos, anyway.

Figures 8-2 and 8-3 illustrate the visual effects of a moderately sized GIF and logo at the top of a page. The *HTML For Dummies* logo file is only 14,000 bytes, so it loads in only a few seconds. The secondary logo is only 5,000 bytes or so. Both images are small and therefore fit easily on display screens with resolution as low as 640 x 480 pixels. Figure 8-2 also illustrates the view with the image loading turned off in Netscape.

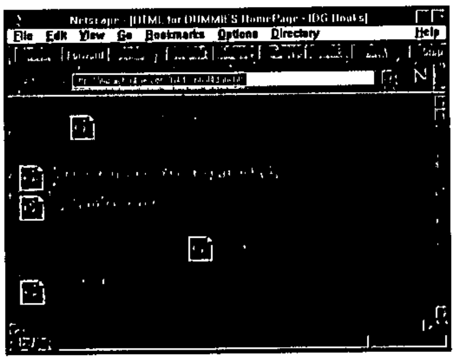


Figure 8-2: HTML document logo viewed with Netscape and image display turned off.

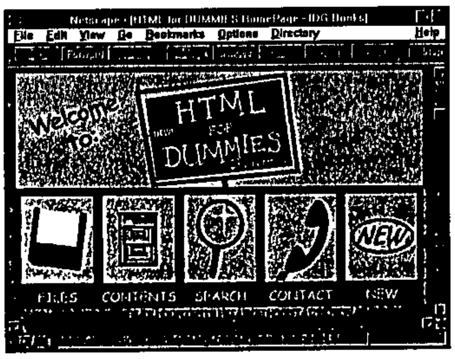
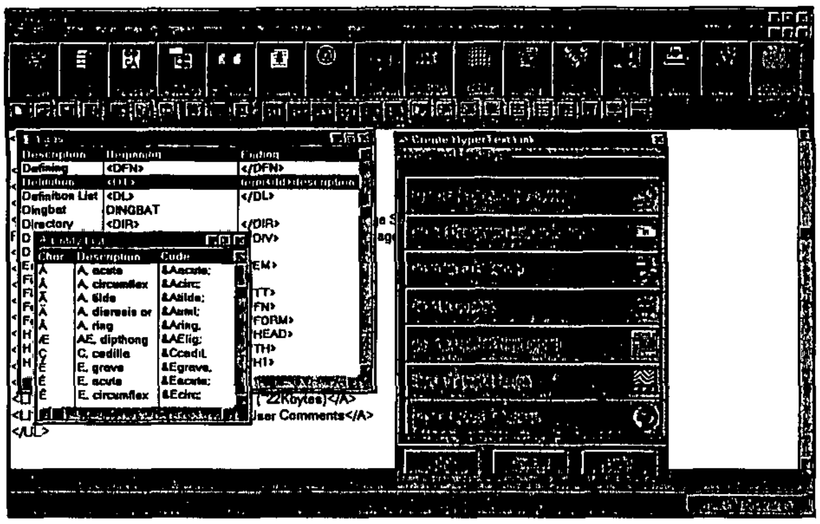


Figure 8-3: HTML document logo viewed with Netscape and images displayed.



Description	Beginning	Ending
Defining	<DFN>	</DFN>
Definition	<DD>	term<DD>description
Definition List	<DL>	</DL>
Dingbat	DINGBAT	
Directory	<DIR>	</DIR>

Char	Description	Code
À	A, acute	&Aacute;
Â	A, circumflex	&Acirc;
Ã	A, tilde	&Atilde;
Ä	A, dieresis or	&Auml;
Å	A, ring	&Aring;
Æ	AE, dipthong	&AElig;
Ç	C, cedille	&Ccedil;
È	E, grave	&Egrave;
É	E, acute	&Eacute;
Ê	E, circumflex	&Ecirc;

Figure 22-5: The HotDog editing screen.

common tags accessible from the standard toolbar makes it easy for the beginner. This can irritate power-users because some useful tags are nearly hidden within the menu system.

InContext Spider (see Figure 22-6) relies on the NCSA Mosaic browser for direct viewing, although any browser can view finished HTML documents. It contains a built-in HTML 2.0 validator. This should be one of the packages you test if you're just starting your Web weaving. An evaluation version of InContext Spider and the complete $99 version are available from InContext Systems at:

http://www.incontext.ca/

Netscape Navigator Gold

Netscape has developed a version of their Navigator browser which has the capability to edit HTML. Navigator Gold is Netscape's answer to standalone HTML editors. Gold incorporates WYSIWYG, drag-and-drop, and the look and feel of the browser you already know and love. Special features include page wizards to simplify the document creation process, a tool chest of templates, and a JavaScript program editor. While we are not big fans of HTML editors, Netscape has done well in creating a two-in-one product that is worth a test drive. You can download the latest version from:

http://www.netscape.com/comprod/products/navigator/gold/
[illegible] index.html

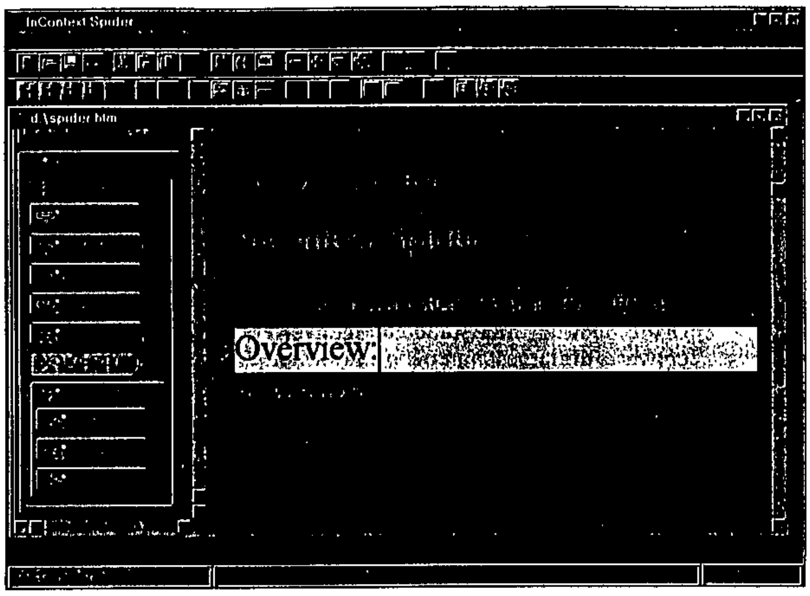


Figure 22-6: The Incontext Spider editing screen.

Word Processor Plug-Ins

If you're joined at the hip to either Word for Windows or WordPerfect for Windows, you may be interested in trying their Internet plug-in packages. These are Word Internet Assistant (Word IA) and WordPerfect Internet Publisher, respectively. They are essentially sets of macros that alter Word and WordPerfect so that they can create and view HTML documents. They provide adequate HTML assistance but aren't really in the same ballpark as the better standalone HTML systems. They're both free at the following locations and at many other mirror sites (a mirror site is a Web or FTP site that copies the contents of another site somewhere else on the Internet, to give users multiple places to download from; most of the really popular Web and FTP sites have mirrors, some have as many as 30 or more).

Internet Assistant for Microsoft Word (6.0 or Word95) is available at:

http://www.microsoft.com/[illegible]

WordPerfect Internet Publisher is available at:

http://[illegible]

character mode When referring to Web browsers, character mode (also called *text mode*) means that such browsers can reproduce text data only. They cannot produce graphics directly without the assistance of a helper application.

clickable map A graphic in an HTML file that has had a pixel coordinate map file created for it, to allow regions of the graphic to point to specific URLs for graphically oriented Web navigation.

client The end-user side of the client/server arrangement, the term *client* typically refers to a consumer of network services of one kind or another. A Web browser is therefore a client program that talks to Web servers.

client/server A model for computing that divides computing into two separate roles, usually connected by a network: The client works on the end-user's side of the connection and manages user interaction and display (input and output, and related processing), while the server works elsewhere on the network and manages data-intensive or shared-processing activities, like serving up the collections of documents and programs that a Web server typically manages.

common controls When designing HTML documents, most experts recommend that you build a set of consistent navigation controls and use them throughout a document (or collection of documents), providing a set of common controls for document navigation.

Common Gateway Interface (CGI) The specification governing how Web browsers can communicate with and request services from Web servers; also the format and syntax for passing information from browsers to servers via forms or document-based queries in HTML.

computing platform A way of referring to the kind of computer someone is using, this term encompasses both hardware (the type of machine, processor, etc.) and software (the operating system and applications) in use.

content For HTML, content is its *raison d'etre;* although form is important, content is why users access Web documents and why they keep coming back for more.

convention An agreed-upon set of rules and approaches that allows systems to communicate with one another and work together.

DARPA (Defense Advanced Research Projects Administration) A U.S. Department of Defense funding agency that supplied the cash and some of the expertise that led to the development of the Internet, among many other interesting things.

dedicated line A telephone line dedicated to the purpose of computerized telecommunications; a dedicated line may be operated continuously (24 hours a day) by its owner. In this book, such lines usually provide a link to an Internet Service Provider.

default In general computer-speak, a default is a selection that's made automatically in a program, instruction, or whatever, when no selections are made explicitly. For HTML the default is the value assigned to an attribute when none is supplied.

desktop (also called *desktop machine*) The computer a user typically has on his or her desktop; a synonym for *end-user computer* or *computer.*

GIF (Graphics Interchange Format) One of a set of commonly used graphics formats within Web documents. It is used frequently because of its compressed format and compact nature.

Gopher A program/protocol developed at the University of Minnesota, Gopher provides for unified, menu-driven presentation of a variety of Internet services, including WAIS, telnet, and FTP.

graphics In HTML documents, graphics are files that belong to one of a restricted family of types (usually GIF or JPEG) that are referenced via URLs for in-line display on Web pages.

grep (general regular expression parser) A standard UNIX program that looks for patterns found in files and reports on their occurrences. The *grep* program handles a wide range of patterns, including so-called "regular expressions," which can use all kinds of substitutions and wildcards to provide powerful search-and-replace operations within files.

GUI (Graphical User Interface) Pronounced *gooey.* GUIs are what make graphical Web browsers possible; they create a visually oriented interface that makes it easy for users to interact with computerized information of all kinds.

heading For HTML, a heading is a markup tag used to add document structure. The term is sometimes be used to refer to the initial portion of an HTML document between the `<HEAD> . . . </HEAD>` tags, where titles and context definitions are commonly supplied.

helper applications Today, browsers can display multiple graphics files (and other kinds of data); sometimes, browsers must pass particular files — for example, motion picture or sound files — over to other applications that know how to render the data they contain. Such programs are called helper applications because they help the browser deliver Web information to users.

hierarchical structure A way of organizing Web pages using links that make some pages subordinate to others. (*See* tree-structure[d] for another description of this kind of organization.)

history list Each time a user accesses the Web, his or her browser normally keeps a list of all the URLs visited during that session; this is called a history list, and provides a handy way to jump back to any page that's already been visited while online. History lists normally disappear when the user exits the browser.

hotlist A Web page that consists of a series of links to other pages, usually annotated with information about what's available on that link. Hotlists act like switchboards to content information, and are usually organized around a particular topic or area of interest.

HTML (HyperText Markup Language) The SGML-derived markup language used to create Web pages. Not quite a programming language, HTML nevertheless provides a rich lexicon and syntax for designing and creating useful hypertext documents for the Web.

http or **HTTP** (hypertext transfer protocol) The Internet protocol used to manage communication between Web clients (browsers) and servers.

MPPP (Multilink Point-to-Point Protocol) An Internet protocol that allows simultaneous use of multiple physical connections between one computer and another, to aggregate their combined bandwidth and create a "larger" virtual link between the two machines.

multimedia A method of combining text, sound, graphics, and full-motion or animated video within a single compound computer document.

MVS (Multiple Virtual Storage). A file system used on IBM mainframes and clones.

navigation In the context of the Web, navigation refers to the use of hyperlinks to move within or between HTML documents and other Web-accessible resources.

navigation bar A way of arranging a series of hypertext links on a single line of a Web page to provide a set of navigation controls for an HTML document or a set of HTML documents.

NCSA (National Center for Supercomputing Applications) A research unit of the University of Illinois at Urbana, where the original Mosaic implementation was built, and where the NCSA *httpd* Web server code is maintained and distributed.

nesting In computer terms, one structure that occurs within another is said to be nested; in HTML, nesting happens most commonly with list structures, which may be freely nested within one another, regardless of type.

netiquette A networking takeoff on the term *etiquette,* netiquette refers to the written and unwritten rules of behavior on the Internet. When in doubt if an activity is permitted or not, ask first, and then act only if no one objects (check the FAQ for a given area, too — it often explicitly states the local rules of netiquette for a newsgroup, mailing list, etc.).

network link The tie that binds a computer to a network; for dial-in Internet users, this is usually a telephone link; for directly attached users, it is whatever kind of technology (Ethernet, token-ring, FDDI, etc.) is in local use.

numeric entity A special markup element that reproduces a particular character from the ISO-Latin-1 character set, a numeric entity takes the form `&#nnn;` where `nnn` is the one-, two-, or three-digit numeric code that corresponds to a particular character (Chapter 6 contains a complete list of these codes).

on-demand connection A dial-up link to a service provider that's available whenever it's needed (on demand, get it?).

online A term that indicates that information, activity, or communications are located on, or taking place in, an electronic, networked computing environment (like the Internet). The opposite of online is *offline,* which is what your computer is as soon as you disconnect from the Internet.

OS (Operating System) The underlying control program on a computer that makes the hardware run and supports the execution of one or more applications. DOS, UNIX, and OS/2 are all examples of operating systems.

packet A basic unit (or package) of data used to describe individual elements of online communications; in other words, data moves across networks like the Internet in packets.

teardown When a network communication session is ending, the two computers agree to stop talking and then systematically break the connection and recover the port addresses and other resources used for the session. This process is called teardown.

technophobe Literally, someone who's afraid of technology, this term is more commonly applied to those who simply want to use technology without understanding it.

telnet The Internet protocol and service that lets you take a smart computer (your own, probably) and make it emulate a dumb terminal over the network. Briefly, telnet is a way of running programs and using capabilities on other computers across the Internet.

template Literally, a model to imitate, we use the term template in this book to describe the skeleton of a Web page, including the HTML for its heading and footer, and any consistent layout and navigation elements for a page or set of pages.

terminal emulation The process of making a full-fledged, stand-alone computer act like a terminal attached to another computer, terminal emulation is the service that telnet provides across the Internet.

test plan The series of steps and elements to be followed in conducting a formal test of software or other computerized systems; we strongly recommend that you write — and use —a test plan as a part of your Web publication process.

text controls Any of a number of HTML tags, including both physical and logical markup, text controls provide a method of managing the way that text appears within an HTML document.

text-mode A method of browser operation that displays characters only. Text-mode browsers cannot display graphics without the assistance of helper applications.

throughput Another measure of communications capability, this term refers to the amount of data that can be "put through" a connection in a given period of time. It differs from bandwidth in being a measure of actual performance, instead of a theoretical maximum for the medium involved.

thumbnail A miniature rendering of a graphical image, used as a link to the full-sized version.

title The text supplied between `<TITLE> . . . </TITLE>` defines the text that shows up on that page's title bar when displayed; it is also used as data in many Web search engines.

token-ring The second most common type of local-area networking technology in use, token-ring is always and forever associated with IBM, because it helped to develop and perfect this type of network. It takes its name from passing around special permits to transmit called *tokens*, in a ring-shaped pattern around the network, to give all attached devices a fair chance to broadcast information whenever they need to.

transparent GIF A specially rendered GIF image will takes on the background color selected in a browser capable of handling such GIFs. This makes the graphic blend into the existing color scheme and provides a more professional-looking page.

tree structure(d) (*see* hierarchical structure) Computer scientists like to think of hierarchies in graphical terms, which make them look like upside-down trees (a single root at the top, multiple branches below). File systems and genealogies are examples of tree-structured organizations that we're all familiar with, but they abound in the computer world. This type of structure also works well for certain Web document sets, especially larger, more complex ones.

UNIX The operating system of choice for the Internet community at large and the Web community, too, UNIX offers the broadest range of tools, utilities, and programming libraries for Web server use.

UNIX shell The name of the command-line program used to manage user-computer interaction, the shell can also be used to write CGI scripts and other kinds of useful programs for UNIX.

URI (Uniform Resource Identifier) Any of a class of objects that identifies resources available to the Web; both URLs and URNs are examples of URIs.

URL (Uniform Resource Locator) The primary naming scheme used to identify Web resources, URLs define the protocols to be used, the domain name of the Web server where a resource resides, the port address to be used for communication, and the directory path to access a named Web file or resource.

URL-encoded text A method for passing information requests and URL specifications to Web servers from browsers, URL encoding replaces spaces with plus signs (+) and substitutes special hex codes for a range of otherwise unreproducible characters. This method is used to pass document queries from browsers to servers (for the details, please consult Chapter 13).

URN (Uniform Resource Name) A permanent, unchanging name for a Web resource, URNs are seldom used in today's Web environment. They do, however, present a method guaranteed to obtain access to a resource, as soon as the URN can be fully resolved (it sometimes consists of human or organizational contact information, instead of resource location data).

Usenet An Internet protocol and service that provides access to a vast array of named newsgroups, where users congregate to exchange information and materials related to specific topics or concerns.

V.32 CCITT standard for a 9.6 Kbps two-wire full duplex modem operating on a regular dial-up or two-wire leased lines.

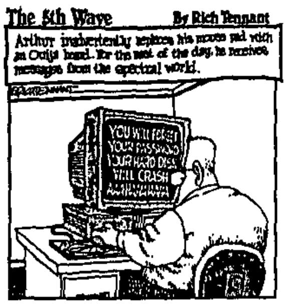
V.32bis Newer higher-speed CCITT standard for full-duplex transmission on two-wire leased and dial-up lines at rates from 4.8 to 14.4 Kbps.

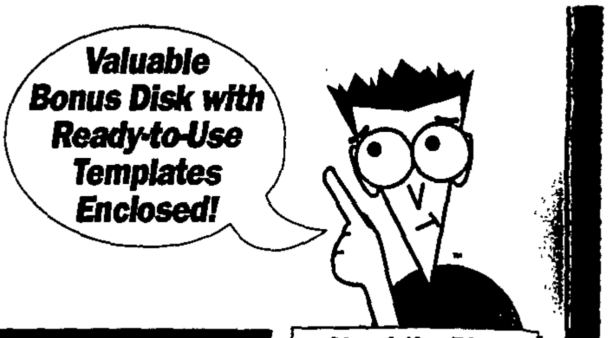
V.34 The newest high-speed CCITT standard for full-duplex transmission on two-wire leased and dial-up lines at rates from 4.8 to 28.8 Kbps.

V.42 CCITT error correction standard that can be used with V.32, V.32bis, and V.34.

Find Out Why Over 25 [illegible]n [illegible] Love the [illegible]mies®

097


The 5th Wave By Rich Tennant
Arthur inadvertently replaces his mouse pad with an Ouija board. For the rest of the day, he receives messages from the spectral world.
YOU WILL FORGET YOUR PASSWORD YOUR HARD DISK WILL CRASH AAAHHHHHHH

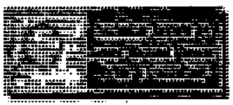


"Good details, clear explanations, and a wonderful layout make this humorous book fun and easy to read."
— Gabby Hon, Indianapolis, IN, on *HTML For Dummies*®

With *HTML For Dummies*®, 2nd Edition, anyone can explore the "ins and outs" of Hypertext Markup Language and gain the know-how to design attractive and informative Web pages in a flash! This friendly reference debunks the Web authoring process and makes it possible for neophytes to build and publish attractive, interesting Web pages with ease.

Use HTML the fun and easy way:

- Discover the latest HTML commands, syntax, and extensions to build Web pages and develop a Net presence
- Use HTML text, images, video, and audio so your pages have impact and flair
- Get a complete overview of the Web's publication process from design to maintenance
- Uncover reviews of Web authoring tools for UNIX, Windows, and the Macintosh



- Apply helpful HTML style and layout tips to design better-looking Web pages
- Plus Ed and Steve's Top Ten Lists:
  - Ten HTML dos and don'ts
  - Ten ways to kill Web bugs
  - Ten considerations to help you decide whether to build or buy your Web service

About the Authors

Ed Tittel is the author of numerous magazine articles and more than 20 computer books. His other IDG Books titles include *Foundations™ of World Wide Web Programming with HTML & CGI*, *The 60-Minute Guide to Java*®, and *NetWare® For Dummies*®, 2nd Edition.

Steve James is a full-time freelance writer, computer researcher, and the coauthor of several computer books.

About the Disk

NEW! The Enclosed Bonus Disk Includes:

- A complete copy of all the materials needed to access the *HTML For Dummies*, 2nd Edition Web site
- Five HTML page templates, ready to be filled out and used
- Files on disk are formatted for PC and Macintosh access (but require the ability to read a PC-formatted 3½" HD diskette)
- All standard HTML tags newer than 2.0 specifications plus Netscape and Internet Explorer extensions!

#1 Bestselling Computer Book Series

READER LEVEL
Beginning to Intermediate

COMPUTER BOOK SHELVING CATEGORY:
[illegible]/Internet/[illegible]Programming/HTML

SYSTEM REQUIREMENTS
[illegible]

```
HTML DUMMIES 2ED
TITTEL, E                 IDGBOO
03IPUB O      Mar/97  133 DISK

1-56884-647-9      $29.99     TP
```

7-9
999

Printed in the U.S.A.

IDG BOOKS WORLDWIDE
FOR DUMMIES COMPUTER BOOK SERIES FROM IDG

IDG Books Worldwide, Inc.
An International Data Group Company
Foster City, CA 9440[illegible]

EASY

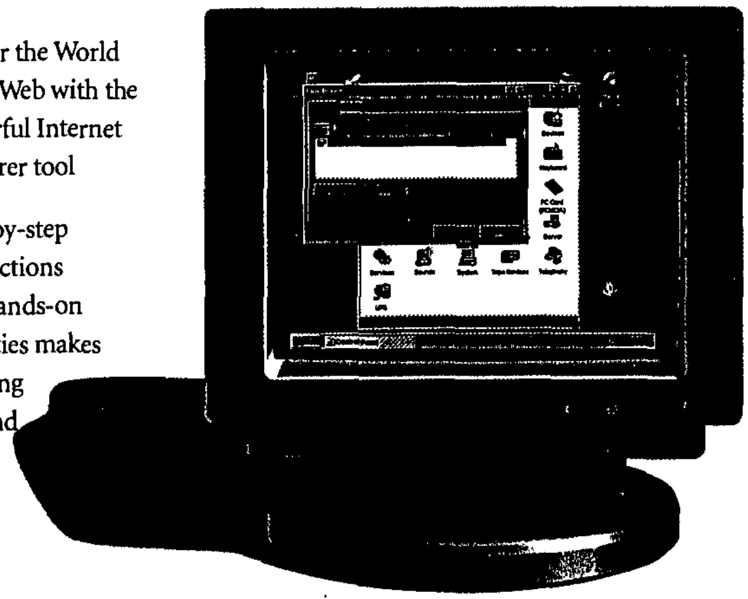
The easy, full-color, step-by-step guide

STEP BY STEP

HOW TO USE MICROSOFT® Windows® NT 4 WORKSTATION

JACQUELYN GAVRON AND JOSEPH MORAN

- Master the World Wide Web with the powerful Internet Explorer tool
- Step-by-step instructions and hands-on activities makes learning fun and easy



- Learn how to make Microsoft's state-of-the-art operating system work for you—even if you've never used Windows before
- Fully updated to cover version 4

FULL COLOR

REAL WORLD EXAMPLES

How to Use

MICROSOFT WINDOWS NT 4 WORKSTATION

JACQUELYN GAVRON AND JOSEPH MORAN

ZIFF-DAVIS
ZD
PRESS

Ziff Davis Press
An imprint of Macmillan Computer Publishing USA
Emeryville, California

[redacted]

Acquisitions Editor	Lysa Lewallen
Copy Editor	Kim Haglund
Technical Reviewer	Ron Pacchiano
Book Design	Dennis Gallagher/Visual Strategies, San Francisco
Technical Illustration	Dave Feasey, Sarah Ishida, and Mina Reimer
Word Processing	Howard Blechman
Page Layout	Martha Barrera
Indexer	Christine Spina

Ziff-Davis Press, ZD Press, and the Ziff-Davis Press logo are trademarks or registered trademarks of, and are licensed to Macmillan Computer Publishing USA by Ziff-Davis Publishing Company, New York, New York.

Ziff-Davis Press imprint books are produced on a Macintosh computer system with the following applications: FrameMaker®, Microsoft® Word, QuarkXPress®, Adobe Illustrator®, Adobe Photoshop®, Adobe Streamline™, MacLink®*Plus*, Aldus® FreeHand™, Collage Plus™.

Screen shots reprinted by permission from Microsoft Corporation.

Ziff-Davis Press, an imprint of
Macmillan Computer Publishing USA
5903 Christie Avenue
Emeryville, CA 94608

THE INFORMATION AND MATERIAL CONTAINED IN THIS BOOK ARE PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY WARRANTY CONCERNING THE ACCURACY, ADEQUACY, OR COMPLETENESS OF SUCH INFORMATION OR MATERIAL OR THE RESULTS TO BE OBTAINED FROM USING SUCH INFORMATION OR MATERIAL. NEITHER MACMILLAN COMPUTER PUBLISHING USA NOR THE AUTHOR SHALL BE RESPONSIBLE FOR ANY CLAIMS ATTRIBUTABLE TO ERRORS, OMISSIONS, OR OTHER INACCURACIES IN THE INFORMATION OR MATERIAL CONTAINED IN THIS BOOK, AND IN NO EVENT SHALL MACMILLAN COMPUTER PUBLISHING USA OR THE AUTHOR BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF SUCH INFORMATION OR MATERIAL.

This book was produced digitally by Macmillan Computer Publishing and manufactured using 100% computer-to-plate technology (filmless process), by Shepard Poorman Communications Corporation, Indianapolis, Indiana.

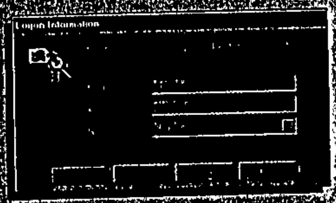
ISBN 1-56276-445-4

Manufactured in the United States of America
10 9 8 7 6 5 4 3 2

INTRODUCTION

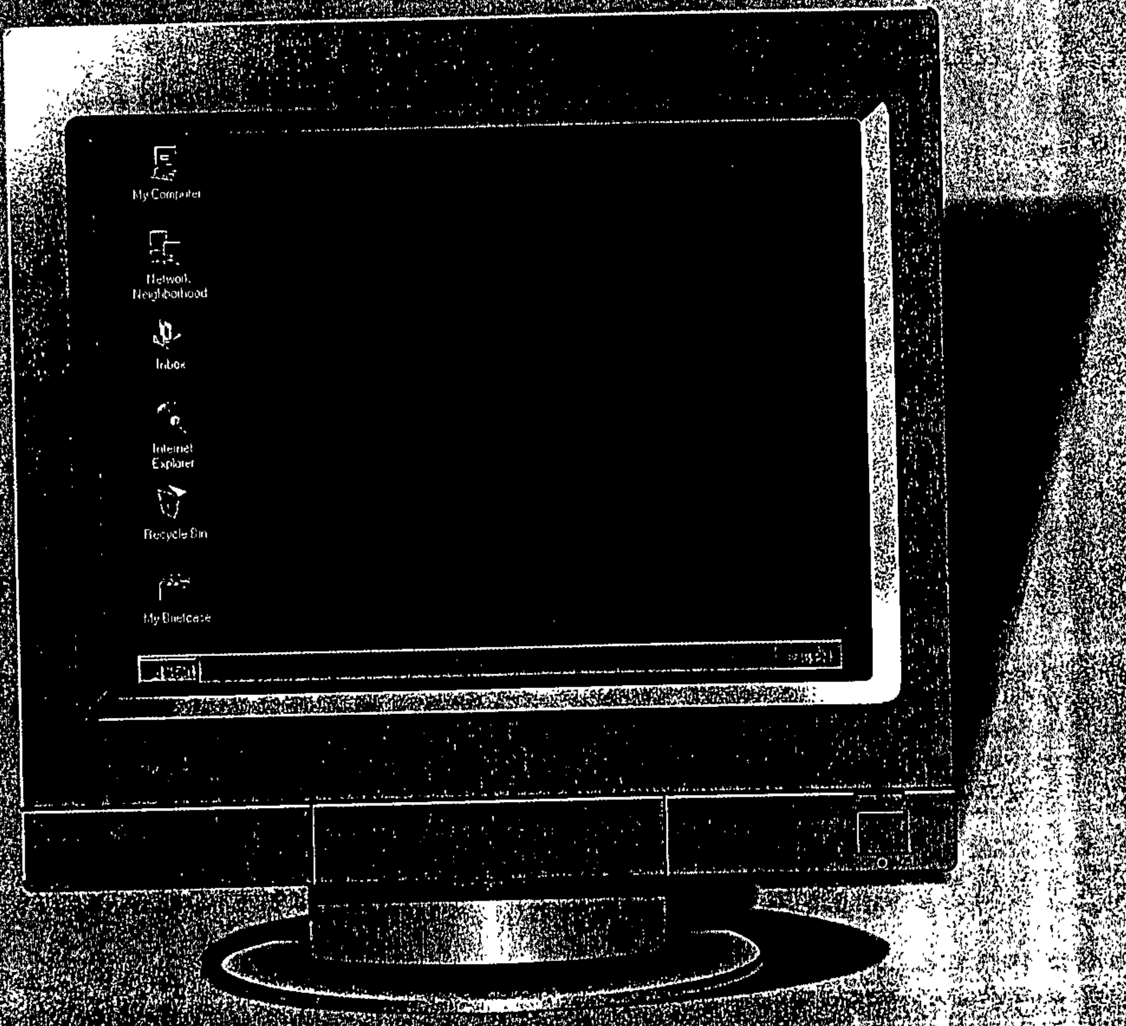
How to Use Microsoft Windows NT 4 Workstation is an illustrated, step-by-step guide to using the newest version of the Windows operating system. Whether you're using Windows for the first time or upgrading to Windows NT 4 from an earlier version of the operating system—Windows 3.1, Windows NT 3.51, or Windows 95—you'll find everything you need to get up to speed with the new operating system as quickly as possible. *How to Use Microsoft Windows NT 4 Workstation* takes a hands-on approach, walking you through each step involved in tasks such as updating documents with My Briefcase, installing and removing applications, and setting up your desktop to look and run the way you like it. Each step is accompanied by screen shots that mirror what you'll see on your computer's monitor.

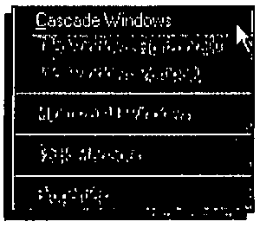
If you're upgrading to Windows NT 4 from Windows 3.1 or NT 3.51, the first thing you'll notice about Windows NT 4 is that it's got a completely different user interface. To introduce you to all these new icons and tools, we begin with the basics: getting started, finding your way around Windows NT, managing files and folders, and working with applications (including those that come with Windows NT). Of course, the look and feel isn't all that's new in Windows NT 4. It's got most everything you need to get on line as well, including a browser called the Internet Explorer. Here you'll find a comprehensive guide to using Windows NT to access the World Wide Web, from setting up your modem to customizing IE and surfing to actual Web sites. Later chapters cover topics ranging from protecting your data and customizing the desktop to networking (which is just one of Windows NT's many strengths). In addition, all chapters contain Tip Sheets with hints and



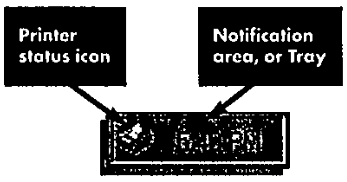
2 Now the Logon Information dialog box appears. In it, you must type both your username and your password. When you're done, click OK. (See the Tip Sheet if you forget your password.)

3 If your username and password are valid, the Windows NT 4 desktop will appear in a few seconds. If not, you'll be presented with an error message that reads "The system could not log you on. Make sure your User name and domain are correct, then type in your password again. Letters in passwords must be typed using the correct case. Make sure that Caps Lock is not accidentally on."

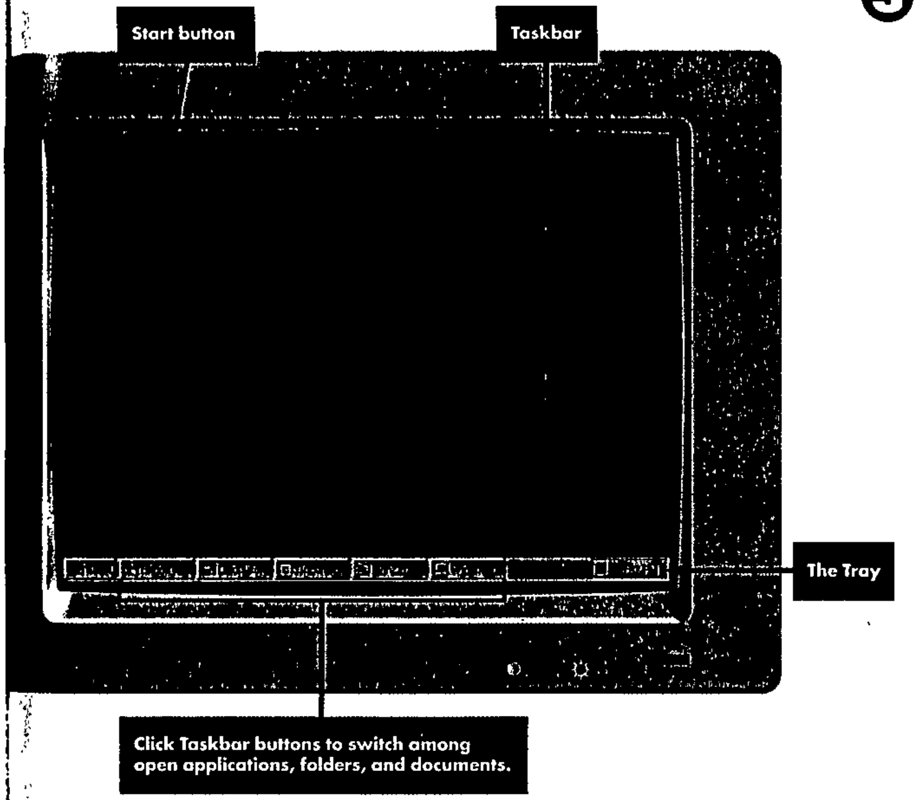


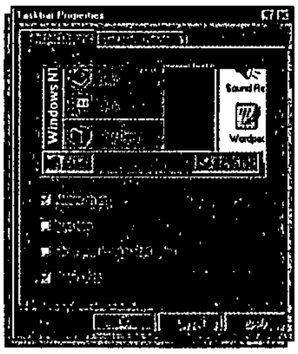


2 You can also use the Taskbar to neatly arrange windows on the screen. Right-click any blank space on the Taskbar and use the menu that appears to arrange windows in one of three ways: overlapping one another (Cascade Windows), stacked (Tile Windows Horizontally), or side-by-side (Tile Windows Vertically).

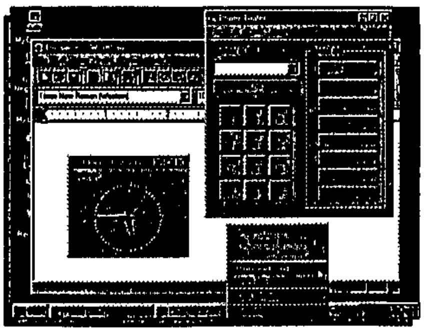


3 To the extreme right of the Taskbar is a notification area called the Tray. It displays information including the time of day and the status of certain peripheral devices such as a modem and a printer.


Start button
Taskbar
The Tray
Click Taskbar buttons to switch among open applications, folders, and documents.



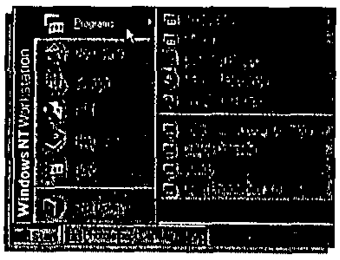
4 To control the way the Taskbar behaves, right-click a blank section and select Properties from the menu that opens. From the Taskbar Options tab, you can specify whether the Taskbar is to remain visible at all times (Always on top). For a less cluttered desktop, you can set the Taskbar so that it appears only when you move the cursor to the bottom of your screen (Auto hide). You can also remove the clock by deselecting the Show Clock option so that the check mark beside it disappears.



5 To clean up the desktop really fast, right-click any blank section of the Taskbar and select Minimize All Windows. Next thing you know, the desktop will be empty, though all applications are still running and accessible from the Taskbar.

How to Launch Applications

When you install applications under Windows NT 4, their icons will appear in the Program's menu that opens when you click the Start button. Selecting a program from this menu is the easiest way to start one up. However, there are several other ways to get the job done: You can create shortcuts on the desktop (which saves a few mouse clicks), double-click an application's icon from within the Windows NT Explorer, or launch it from the command line. Here, you'll learn how to launch applications three ways, and later in this chapter, you'll learn how to launch them using shortcuts. When you finish this chapter, you'll have a whole slew of techniques for starting applications. The choice is yours.

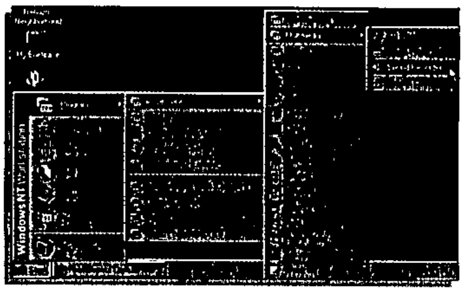


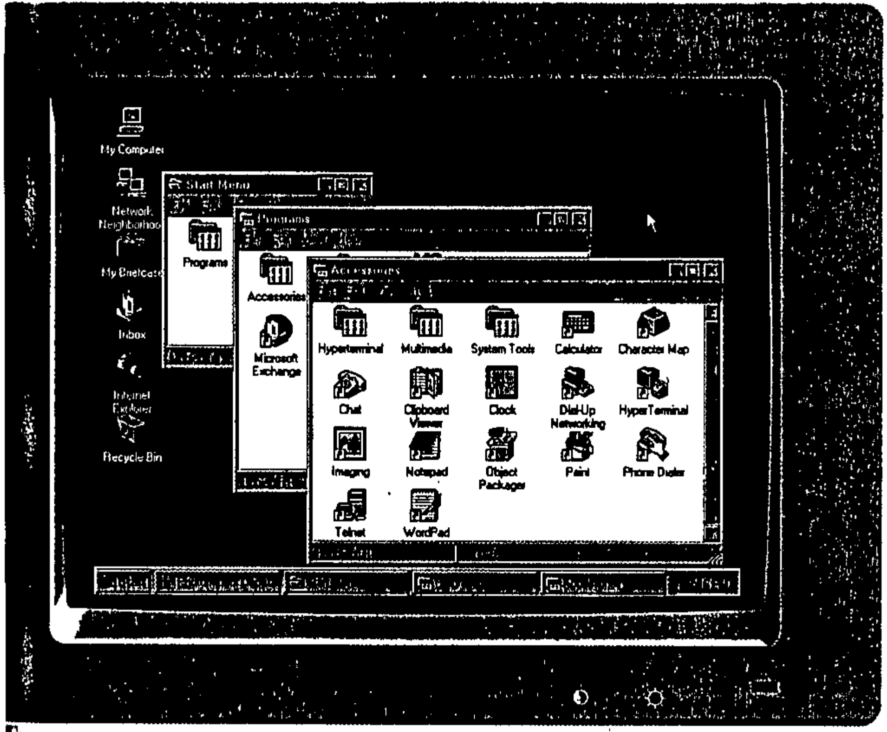
▶ 1 The easiest way to launch an application is from the Start menu. Click the Start button, then choose Programs. The menu that opens contains both individual applications (like Windows NT Explorer) and program groups (like Accessories). To launch Windows NT Explorer, just click on it.

TIP SHEET

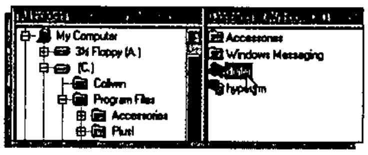
- **To open the Start menu using the keyboard, press Ctrl+Esc. Once it's open, use the arrow keys to navigate, or else click the underlined letter to select an item (for example, press D to open the Documents menu).**
- **Techies take note: If you type the name of a 16-bit Windows application in the Start menu's Run dialog box, Windows NT gives you the option to run the application in separate memory space. The benefit: If this application crashes, other programs won't be affected. The catch: This method consumes more memory, so if you're low, avoid it.**

2 If, however, you want to launch a program that resides within a program group, glide the mouse over the item. All programs within that group then appear. Sometimes, there are even groups within groups. Accessories, for instance, contains several other program groups: Hyperterminal, Multimedia, and System Tools. (You can always tell when there are subgroups because a black arrow appears beside the item.)



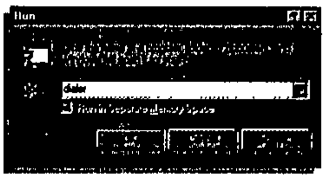


3 As you learned in step 2, Windows NT 4 often nests programs deep within the Start menu. If you use an application often, all those mouse clicks get tiresome. That's where shortcuts come in. To learn how to create a shortcut to launch an application, see "How to Create Shortcuts to Applications" later in this chapter.



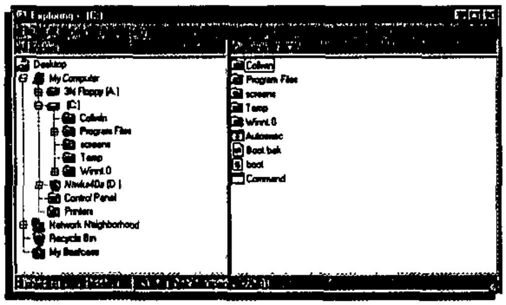
4 Another way to launch applications is by using the Windows NT Explorer. Open it, find the program you want to launch, and then double-click it.

5 Finally, you can also launch applications using the Start menu's Run command. Just click the Start button, choose Run, and either type in the name of the program, or click the Browse button to locate it.



How to Create Shortcuts to Applications

Shortcuts are one of the most useful features in Windows NT 4 because they can really save time (and mouse clicks). A shortcut allows you to quickly launch applications directly from the desktop, rather than wading through the Start menu or the Windows NT Explorer. In fact, you can create a shortcut to just about any object, including applications, printers, disk drives, and folders. There are several ways to create shortcuts. Here, you'll learn the easiest and most reliable methods.

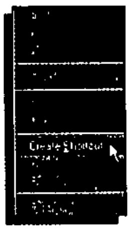


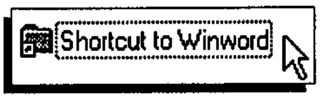
1 To create a shortcut to a document or an application, first open Windows NT Explorer.

TIP SHEET

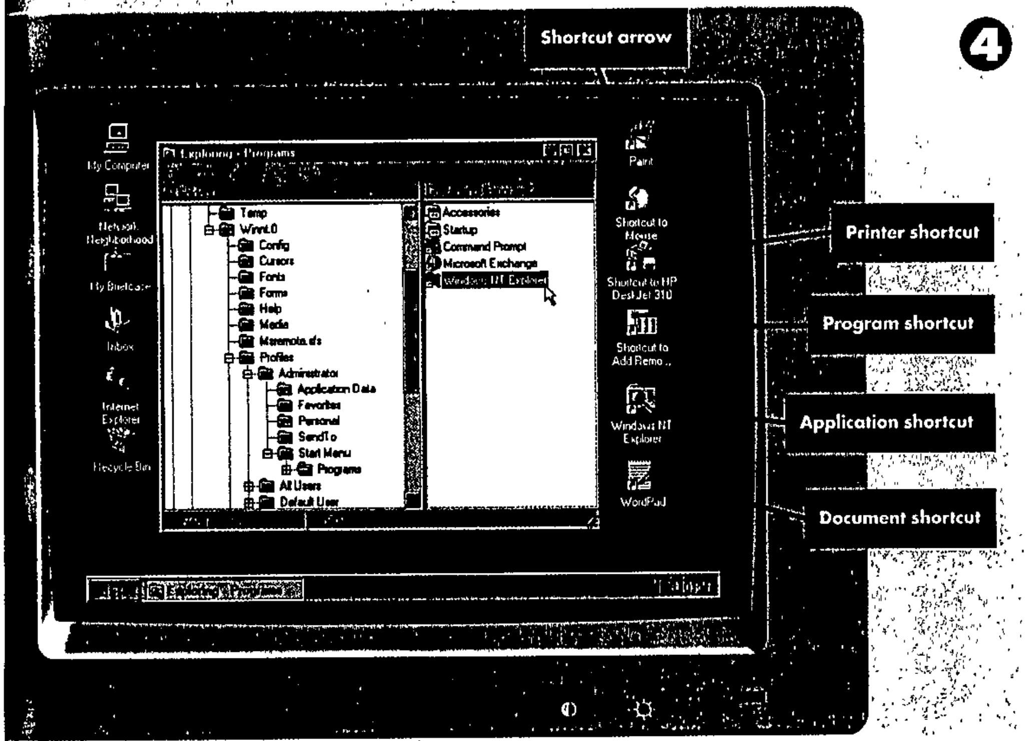
- To locate the file or document to which you want to create a shortcut, use the Find utility on the Start menu. Or, pull down the Windows NT Explorer's Tools menu, then choose Find.
- Another way to create a shortcut to any object is to use the right mouse button to drag it to the desktop (or to another folder). Then choose Create Shortcut(s) from the menu that pops open.
- Another way to delete a shortcut is to drag it to the Recycle Bin.
- To create a shortcut in a folder, open Windows NT Explorer's File menu. Choose New, then Shortcut. You're then prompted to type the location of the item you want a shortcut to, and a name for the shortcut.

❷ Locate the application or document you need (see the Tip Sheet below). When you've found it, right click its icon and select Create Shortcut from the menu that appears.

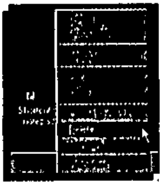




❸ A shortcut to the object will now appear in the Explorer's right pane.



❹ Now drag the shortcut from the Windows NT explorer onto the Desktop.

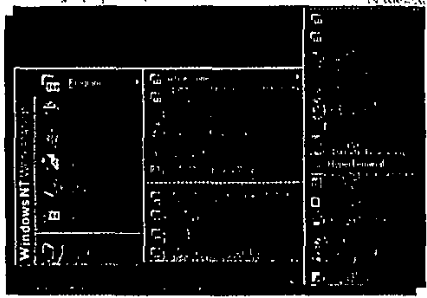


❺ To remove a shortcut from the desktop, right-click the shortcut. Then select Delete from the context menu. Alternatively, you can simply drag the shortcut onto the Recycle Bin.

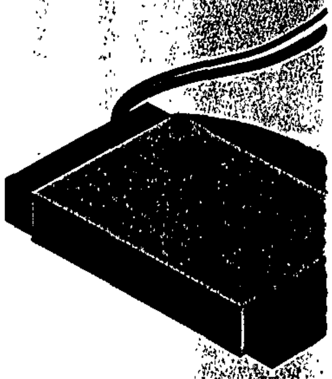
❻ To rename a shortcut, select the icon, press F2, then type in the new name. (Alternatively, you can right-click the shortcut and choose Rename.)

How to Use HyperTerminal

HyperTerminal is a communications application that comes with Windows NT. It's particularly useful for connecting directly to bulletin board systems (BBSs) or to other computers—to transfer files, for example. Although most online communications takes place on the Internet or a commercial online service such as America Online or CompuServe, you may still need to connect to a good old-fashioned BBS now and then.

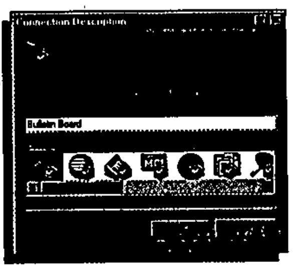


▶ 1 Click the Start button, then choose Programs, Accessories, then HyperTerminal.



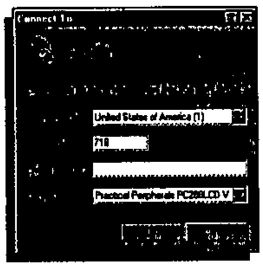
TIP SHEET

- ▶ To change the font in which the BBS text appears, pull down the View menu, then choose Font. A dialog box opens in which you can select the font type and size. There's also a preview window so you can see what a font looks like before applying it.



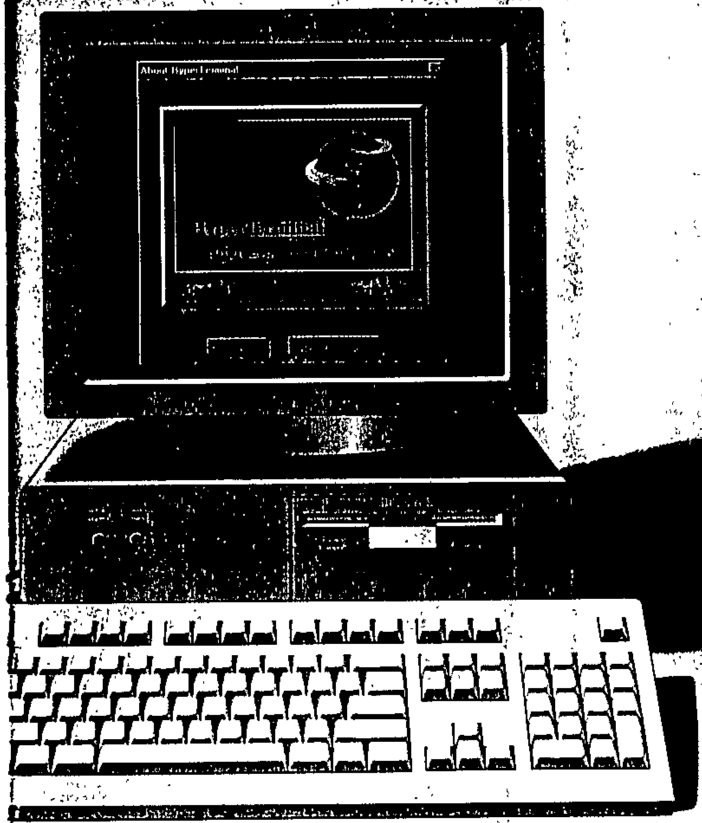
Bulletin Board

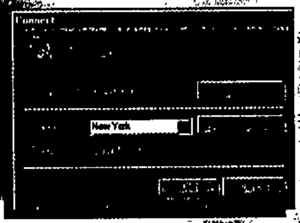
2 In the Connection Description dialog box, enter the name of the computer you will be connecting to. You can also choose an icon to launch this connection in subsequent sessions. Click OK and the Connect To windows pops open.



United States of America (1)
718
Practical Peripherals PC288LCD V

3 You'll notice that upon opening, the Connect To window already contains your area code and country—that's because you entered this information when you set up the modem. Now the only information you need to enter is the phone number of the computer you want to connect to. Then press OK and the Connect window opens.





New York

4 Here, press the Dial button. HyperTerminal dials the number for you and waits for a response from the other computer. While placing the call, HyperTerminal pops open a window that displays the status of the call, such as whether it's dialing or has encountered a busy signal. When the remote computer answers, it displays a welcome or log on message.

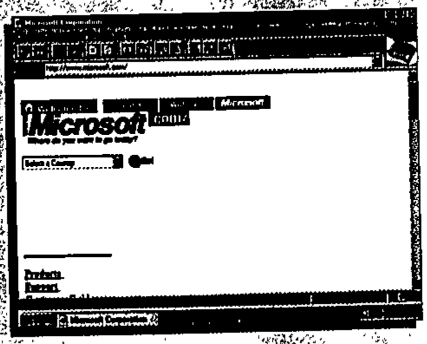
How to Use the Internet Explorer

Windows NT 4 comes with a built-in browser, called Internet Explorer, that you can use to surf the World Wide Web, or the Web, for short. The Internet Explorer's icon appears on the desktop when you install Windows NT. Here, you'll learn the basics of connecting to the Web and using Internet Explorer (IE). One caveat, though: To complete the steps in this chapter, you must already have an account with an Internet service provider or else be able to connect to the Web via your company's local area network.

TIP SHEET

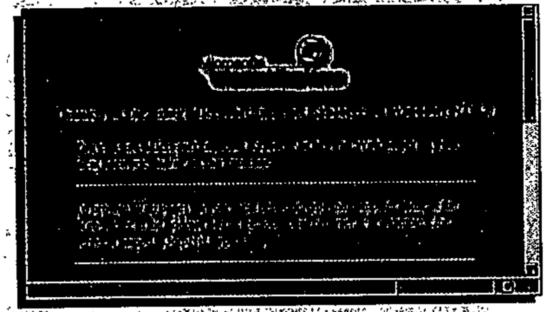
- **Multimedia elements on Web pages are slow to download. If you like, you can configure IE so it doesn't display pictures or animations, or play sounds. To do this, from the View menu choose Options. Click the Appearance tab, and deselect the picture, animation, and sound options.**
- **To print a Web page, choose Print from the File menu. Or, click the printer icon on IE's toolbar. Remember: If you choose not to show pictures in the Options dialog box, they won't appear When you print the page. All you'll see is a placeholder icon.**
- **To search for a word on a Web page and avoid scrolling, from the IE Edit menu click Find. Then type in the word or phrase you want to search for.**

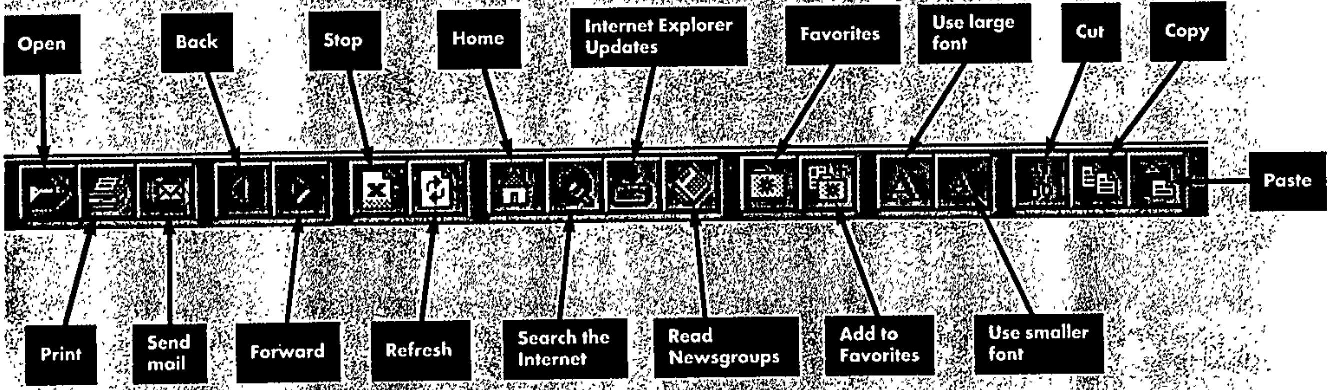
1 Double-click the Internet Explorer (IE) icon on the desktop. (The graphic shown on the next page identifies all the buttons you'll see on IE's toolbar.)

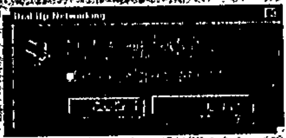


5 Either way, once you're on the World Wide Web and at Microsoft's Web site be patient: The page can take a few seconds to load. (The rate at which a page loads depends on the speed of your modem.) For an in-depth romp around the Internet, see the Try It! following this chapter.

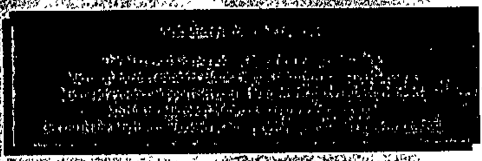
2 The first time you launch Internet Explorer, this Start Page appears. In fact, it will display each and every time you open the browser. (Later in this chapter, though, you'll learn how to replace this plain-vanilla page with one that you choose.)







4 If you're accessing the Web through your company's LAN, clicking www.microsoft.com brings you directly to the site. If, however, you're accessing the Web through a dial-up networking connection, clicking www.microsoft.com opens a dialog box that asks if you want to dial a remote network. Click the "Yes, dial" button.



3 Scroll to the bottom of the page until you see the *hyperlink*, shown here, that will take you to Microsoft's Web site. Now click on the link that reads http://www.microsoft.com. (All URLs begin with "http://," but from now on we will omit this prefix when referring to hyperlinks.)

How to Change the Start Page

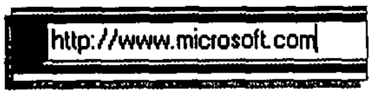
Internet Explorer lets you select a Start page, which is the page that appears each time you launch IE. The default Start page contains basic information on using Internet Explorer that's useful the first few times you launch IE. But after that, you won't pay it any mind. Why not replace it with a screen that does grab you? For example, if there's a page on the Web that you access frequently, you can have it come up automatically each time you start Internet Explorer—you'll learn how to do that here.

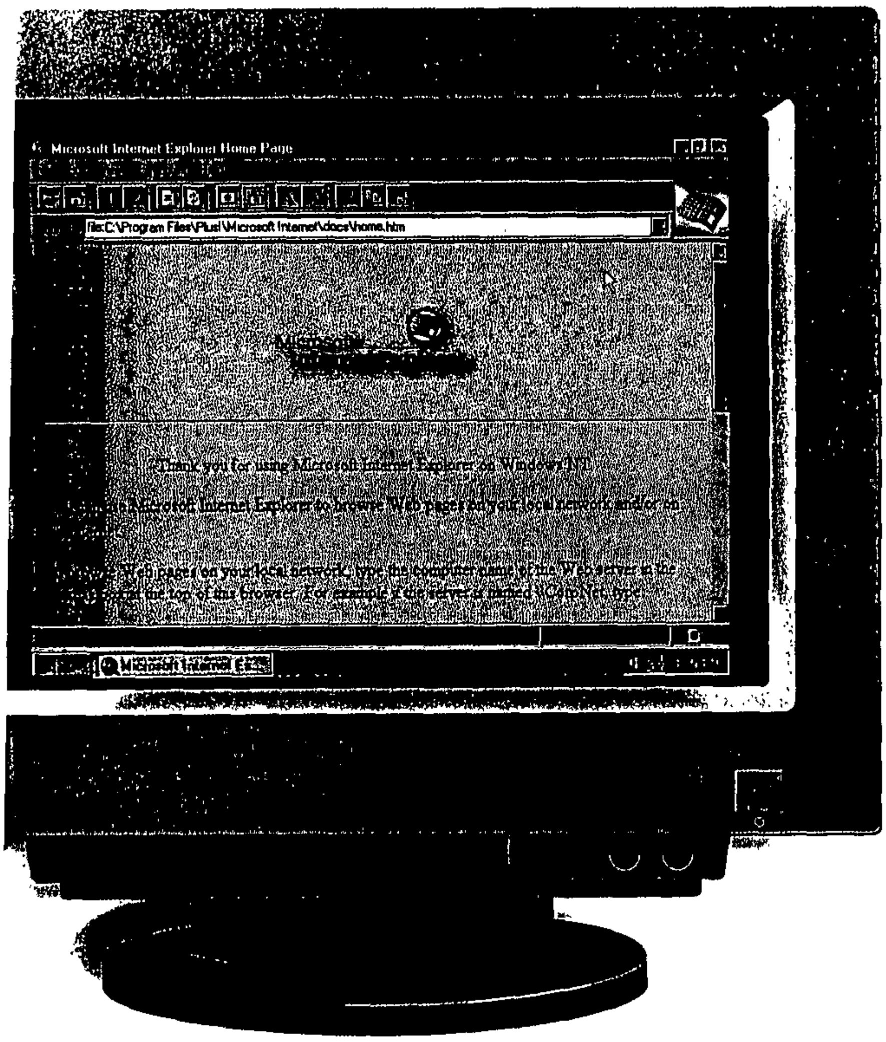
▶ 1 Double-click the Internet Explorer (IE) icon on the desktop.

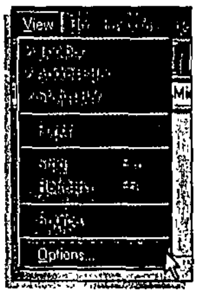
TIP SHEET

- **If you grow nostalgic for the Internet Explorer's Start page, you can restore it. From the View menu, choose Options, then the Start Page tab. Now press the button called Use Default.**
- **To see if there's a new version of Internet Explorer, go to Microsoft's Web site at www.microsoft.com. From there, you can download new versions for free.**

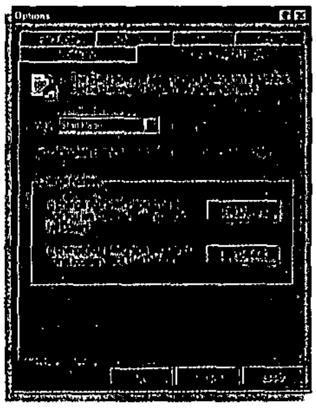
2 In the Address field, type the URL of the page you want to use as the Start Page. For example, here we're using www.microsoft.com as the Start page. Remember: Although most URLs begin with "http://" (for hypertext transport protocol) you don't need to type all this in when entering an address. Forget about the "http://" and begin typing with the part of the address that reads "www."




Microsoft Internet Explorer Home Page
file:C:\Program Files\Plus!\Microsoft Internet\docs\home.htm
Thank you for using Microsoft Internet Explorer on Windows NT.
Microsoft Internet Explorer



3 From the View menu, select Options. Doing so opens the Options dialog box.

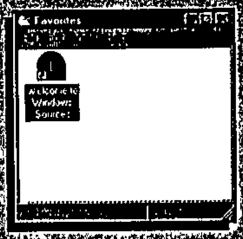


4 Now click the Start and Search Pages tab in the Options dialog box. In the Change Address area, click the button labeled Use Current. This instructs Internet Explorer to automatically load the current Web page each time it launches.

How to Save Favorite Web Sites

Internet Explorer has a feature called Favorites that you can use to save the addresses of the Web sites you visit regularly. This spares you from manually typing in an URL each time you want to surf to a site, letting you access sites easily and quickly.

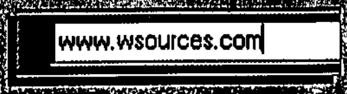
1 Double-click the Internet Explorer (IE) icon on the desktop.



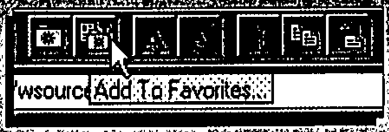
6 To launch a favorite page, just click on its icon in the Favorites folder.

TIP SHEET

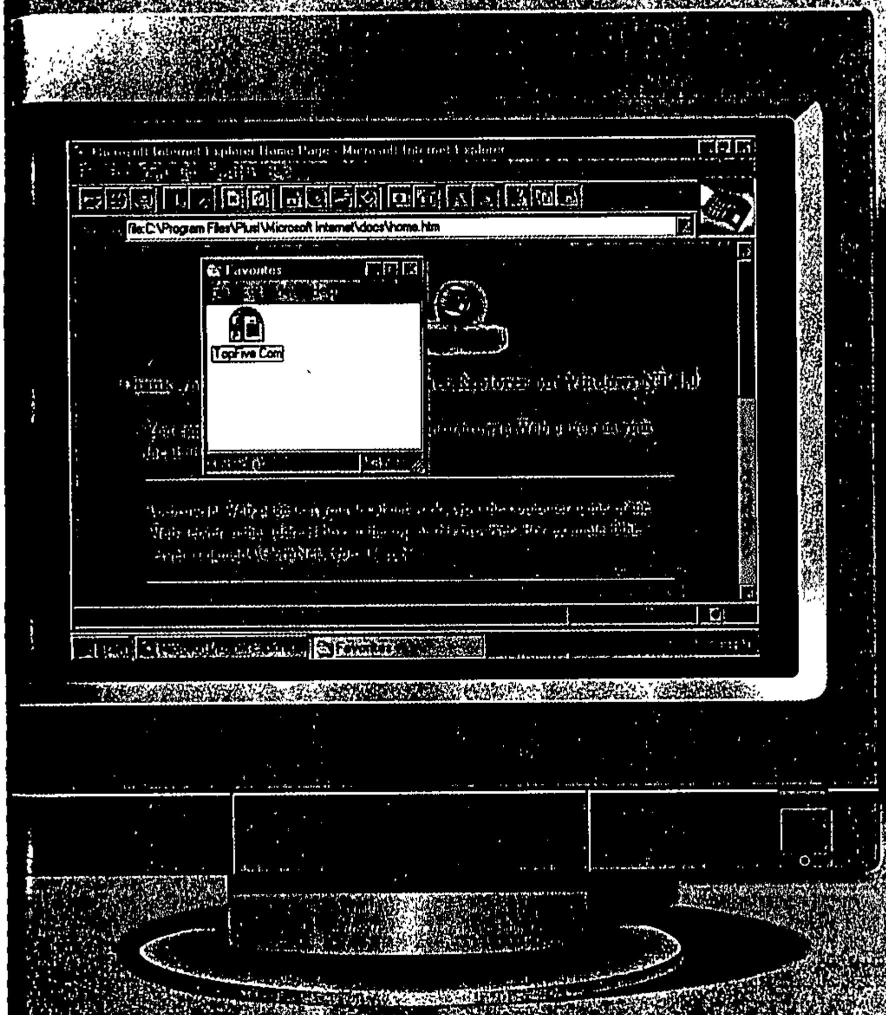
- To add additional Web sites to your Favorites list, repeat steps 2 through 4. You can havo as many sites in Favorites as you like.
- Another way to access a Web site is by typing its URL in the Start menu's Run dialog box.

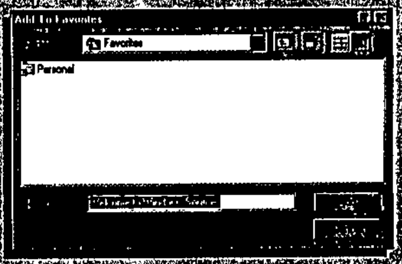


2 In the Address field, enter the URL of the Web site you want to add to your Favorites list. Then press Enter to surf to that site.

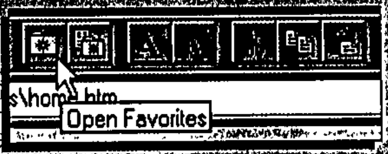


3 After the page loads, click the Add to Favorites button on the toolbar. Doing so opens the Add to Favorites window.


Microsoft Internet Explorer Home Page - Microsoft Internet Explorer
file:C:\Program Files\Plus!\Microsoft Internet\docs\home.htm
Favorites
TopFive Com
Favorites



4 When you open the Add to Favorites window, notice that IE has already named the page for you. You can either keep this name or change it by typing in a new one. When you're done, click Add and Internet Explorer adds that Web page to your Favorites folder.

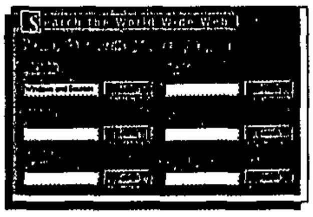


5 To access your Favorites folder, click the Open Favorites button on Internet Explorer's toolbar.

How to Search the Internet

There are millions of Web sites on the Internet and finding the information you need isn't always easy. *Search engines,* such as AltaVista and Yahoo, can help. These tools index or categorize the contents of Web servers and filter information based on search criteria you enter. You can access all of the popular search engines by using the Internet Explorer's Search the Internet tool. Here's how it works.

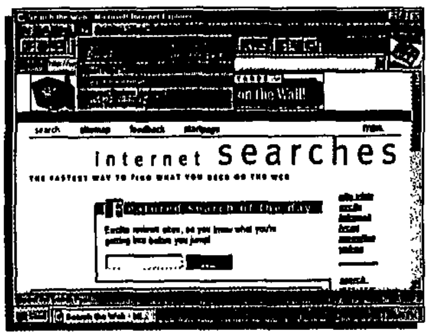
1 Double-click the Internet Explorer icon. Your Start page loads automatically.


Search the World Wide Web

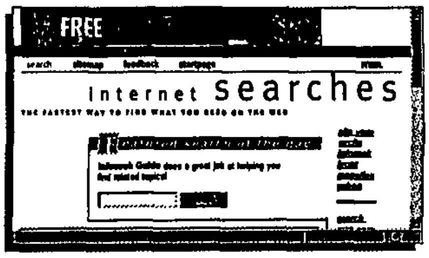
7 If you don't find the results you want, try another search engine. Internet Explorer's search page lets you try them all.

TIP SHEET

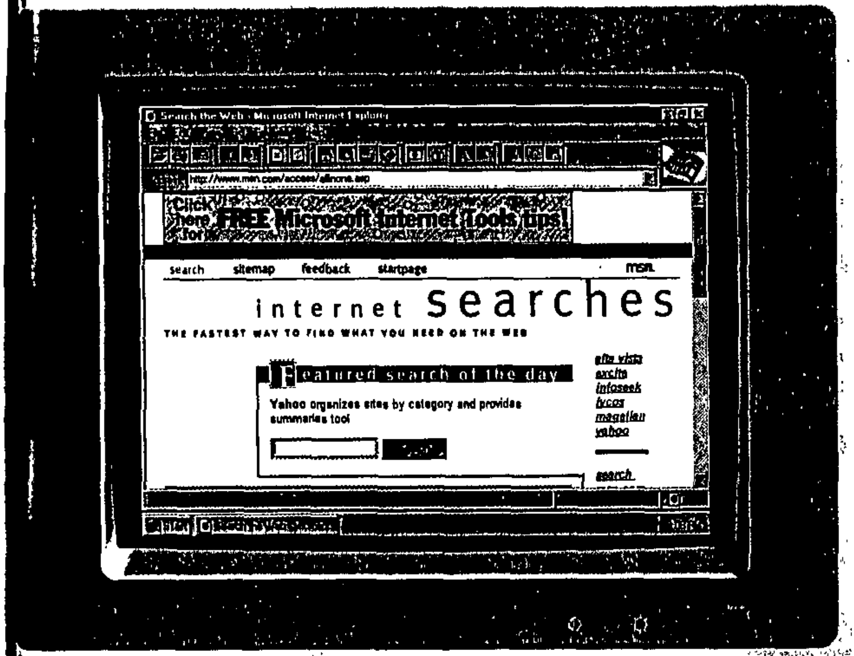
- **For the best results, use boolean operators such as "and" and "or" in your search criteria. They narrow the search so you don't get a million hits. For example, when searching for Windows Sources, we entered "Windows and Sources," which produced 11 hits. However, entering "Windows Sources" yielded over 3 million hits, because Yahoo displays every site containing the word "windows" or "sources." The "and" operator tells the engine to search for both words as one phrase.**
- **To replace the Internet Explorer's search page with another (such as Yahoo or AltaVista), first surf to the search page you want to use. From the View menu choose Options. Next, click on the Start and Search Pages tab, select Search Page from the drop-down menu, and click the Use Current button.**

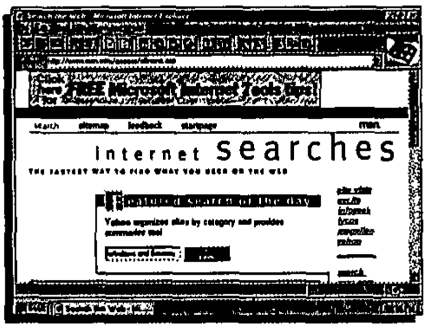


2 To search the Internet, choose Search the Internet from the Go menu. (Or, click on the Search the Internet icon.)

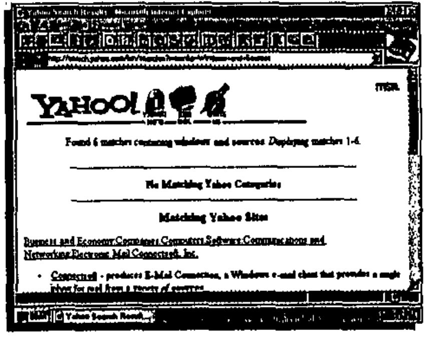


3 Now you'll see the Microsoft Network search page. It contains links to all of the popular search engines and features a different one each time you perform a search.


Search the Web - Microsoft Internet Explorer
http://www.msn.com/access/allinone.asp
Click here FREE Microsoft Internet Tools tips! for
search
sitemap
feedback
startpage
msn
internet searches
THE FASTEST WAY TO FIND WHAT YOU NEED ON THE WEB
featured search of the day
Yahoo organizes sites by category and provides summaries tool
alta vista
excite
infoseek
lycos
magellan
yahoo
search


Click here FREE Microsoft Internet Tools tips!
search
sitemap
feedback
startpage
msn
internet searches
THE FASTEST WAY TO FIND WHAT YOU NEED ON THE WEB
featured search of the day
Yahoo organizes sites by category and provides summaries tool
search

4 In the blank field, type in the search criteria, then press the Search button. To search for Windows Sources magazine, we entered Windows and Sources (see the Tip Sheet for details).



Found 6 matches containing windows and sources Displaying matches 1-6

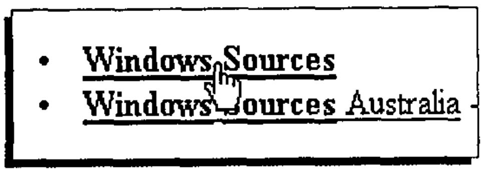
No Matching Yahoo Categories

Matching Yahoo Sites

Business and Economy:Companies:Computers:Software:Communications and Networking:Electronic Mail:Connectsoft, Inc.

- Connectsoft - produces E-Mail Connection, a Windows e-mail client that provides a single place for mail from a variety of sources.

5 A results window then displays the number of *hits*–that is, the items that matched the search criteria you entered.



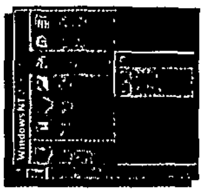
- Windows Sources
- Windows Sources Australia

6 Scroll the window to find the item you want. Then move the cursor over it; the cursor turns into a hand. Now click on the link and you automatically jump to that site.

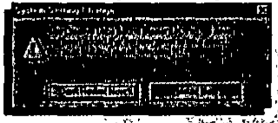
How to Play Audio Files on the Web

Many Web sites, including www.pcweek.com and www.msnbc.com, offer more than text and graphics; you can also play audio files. PCWeek, for example, has a feature called PCWeek Radio where you can hear reporters covering computer industry news.

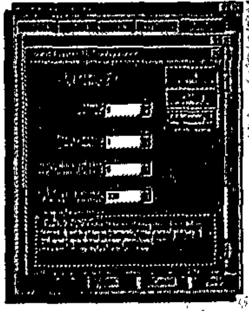
To enjoy audio on the Web, you've got to have a sound card installed. (You need speakers, too, of course.) That's what you'll learn to do here. But before getting started, you must put the sound card into a free expansion slot in your computer, and jot down the resources (such as the IRQ and DMA Channel) that the card uses. Typically, this information is in the documentation or on the sound card itself.



❶ Click the Start button, then Settings, and Control Panel.



❽ The System Setting Change window opens; click Restart Now. When Windows NT restarts, your sound card should be working and you can enjoy audio on the Internet.

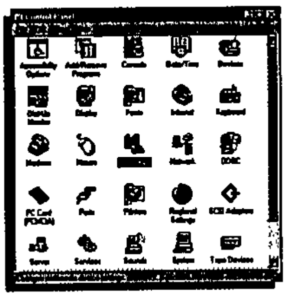


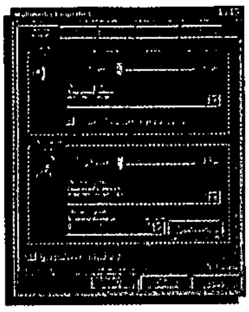
❼ In the Sound Blaster 16 Configuration box that appears, you select the Iinterrupt, the DMA Cchannel, and the 16-bit DMA Cchannel, and the MPU401I/O Aaddress. Use the drop- down menus to enter the appropriate values for each. Then click OK. (The options that appear on the pull- down menus vary by card; the ones you see here are for the Sound Blaster 16 card we used.)

TIP SHEET

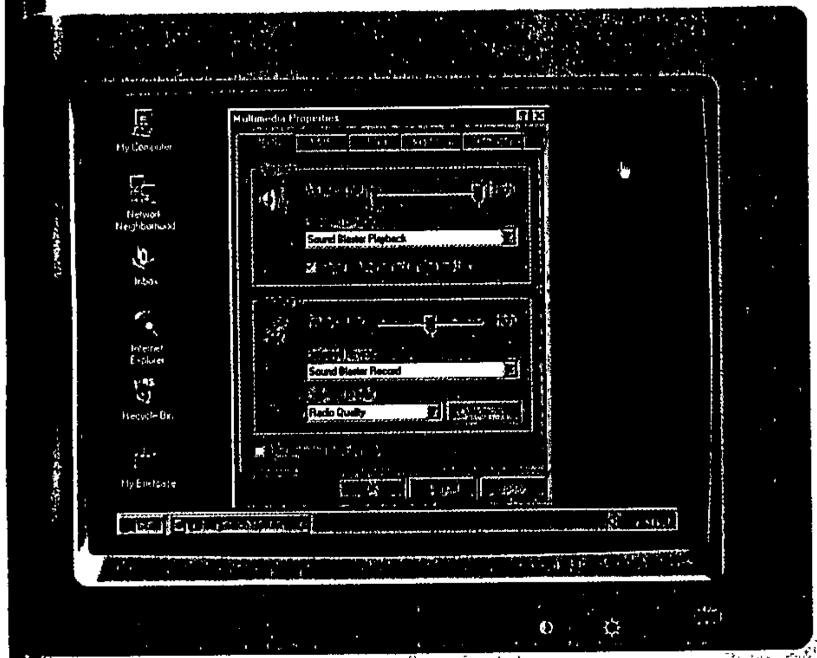
- To play audio files, most Web sites require you to download a special applet called a *player*. PCWeek, for instance, *uses* the RealAudio Player.

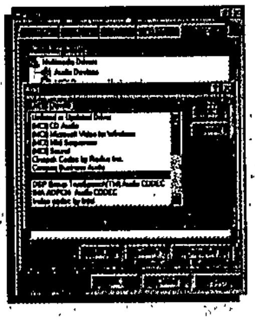
2 Double-click the Multimedia icon, which is used to configure devices such as sound cards and CD-ROMS.



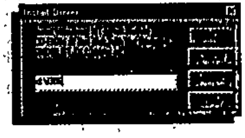


3 In the Multimedia Properties window, click the Devices tab—it's the last one on the right.

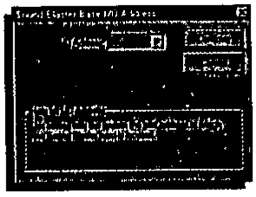




4 Now click the Add button. It lists the multimedia drivers that Windows NT 4 comes with. Find your sound card's name in the list, and double-click it. (If your sound card isn't on the list, click Unlisted or Updated Driver at the top of the list.)



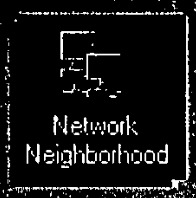
5 The Install Driver window pops up and asks for the Windows NT CD-ROM. (If your device is not on the list, insert the disk that came with your sound card; it contains the driver.) Type in the path (usually d:/i386), and click OK.



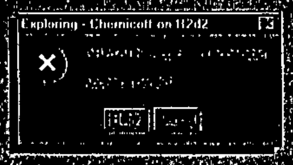
6 Now the fun starts. Because Windows NT 4 doesn't support Plug and Play, you must specifiy what resources the sound card will use. So, in the Sound Blaster Bbase I/O Address, go to the I/O Address area and pick your card's address from the drop-down list. Then click Continue.

How to Access a Computer on the Network

Just as My Computer displays the drives connected to your PC, the Network Neighborhood displays all the other computers connected to your network. Using Network Neighborhood you can explore the content of these computers, such as files, folders, even printers.



1 Double-click the Network Neighborhood icon on the desktop.



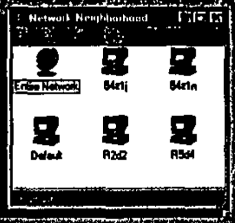
Exploring - Chemicoff on H2d2

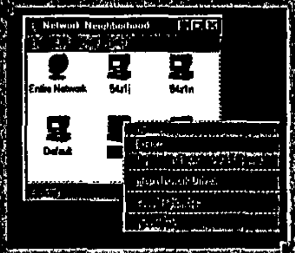
6 If you try to open a folder you're not authorized to use, a dialog box appears telling you that access is denied. Click Cancel to get rid of the window.

TIP SHEET

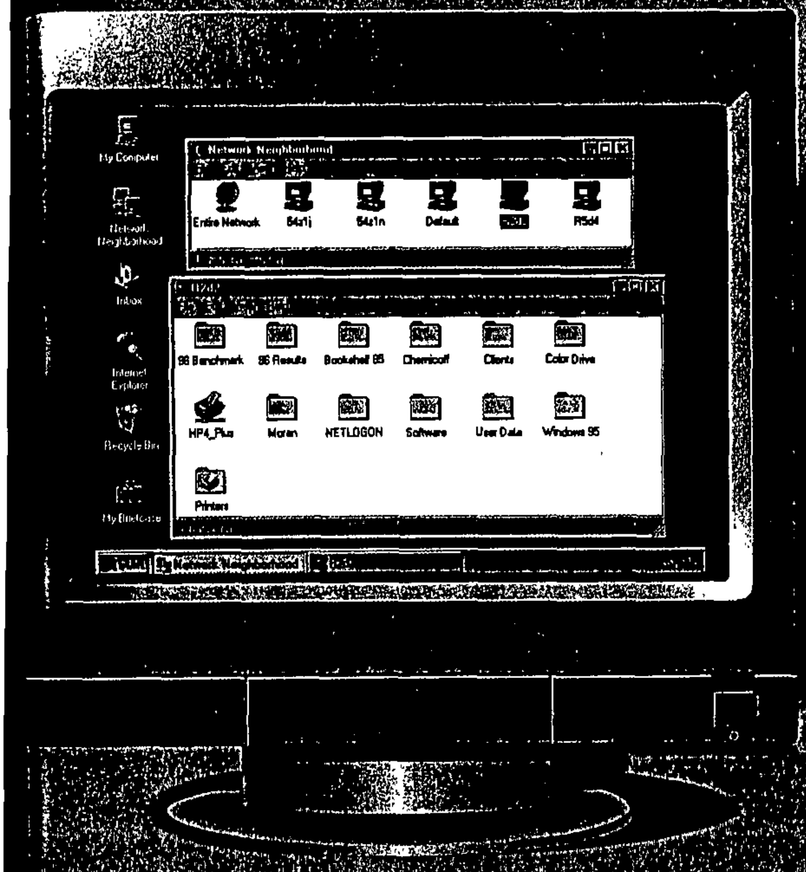
- **To access a machine that's on the network but not part of your workgroup or domain, double-click the Entire Network icon.**
- **To identify whether an icon represents a workstation or a server, right-click it, and then choose Properties. In the General tab, look in the Type area. If the machine name includes the word "Primary" or "backup," it's a server. Individual computers, in turn, are simply called workstations.**
- **To access a network computer without launching the Network Neighborhood, just create a shortcut to it. First, use the right mouse button to drag the computer's icon to the desktop. Then choose Create Shortcut(s) Here from the menu that opens. A shortcut to the server now appears on the desktop.**

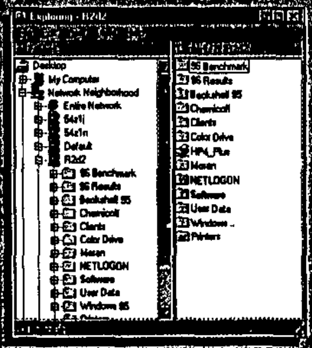
2 The Network Neighborhood window opens. It contains icons representing all the computers included in your workgroup or domain.





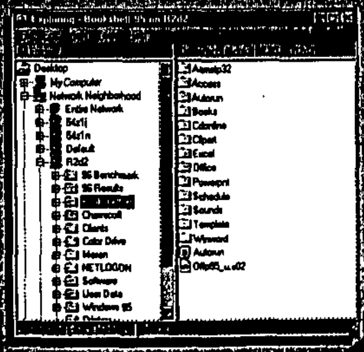
3 To examine the contents of a networked computer, right-click its icon and choose Explore from the menu that opens.





4 The Exploring window that opens displays icons for all computers on the network. However, the computer you selected is expanded one level; that is, all the shared resources it contains, such as folders and printers, are displayed.

5 To view the contents of a particular folder, double-click it. All the folder's contents then appear in the right pane.

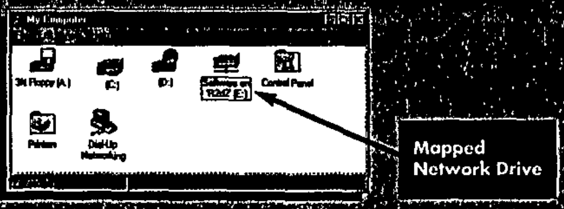


How to Add a Network Drive

On the previous page, you learned how to access resources, such as files and folders, that reside on another computer on the network. If you frequently use this resource, you can save yourself a lot of time by *mapping* the drive to your computer.

When you map a network drive, NT assigns it a unique drive letter and can automatically connect to it when you log on; this saves you the trouble of repeatedly navigating the network to find a particular folder.

1 Double-click the Network Neighborhood icon on the Windows NT desktop.

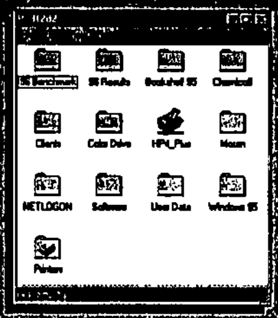


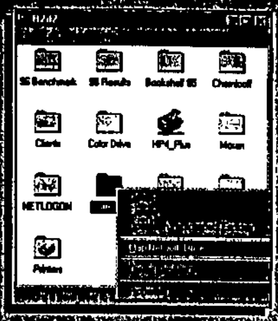
8 The last step is to double-click the My Computer icon. Inside, you'll notice that the icon representing the mapped drive looks different than the hard disk icon representing drive C. This indicates that the mapped drive is connected to the network.

TIP SHEET

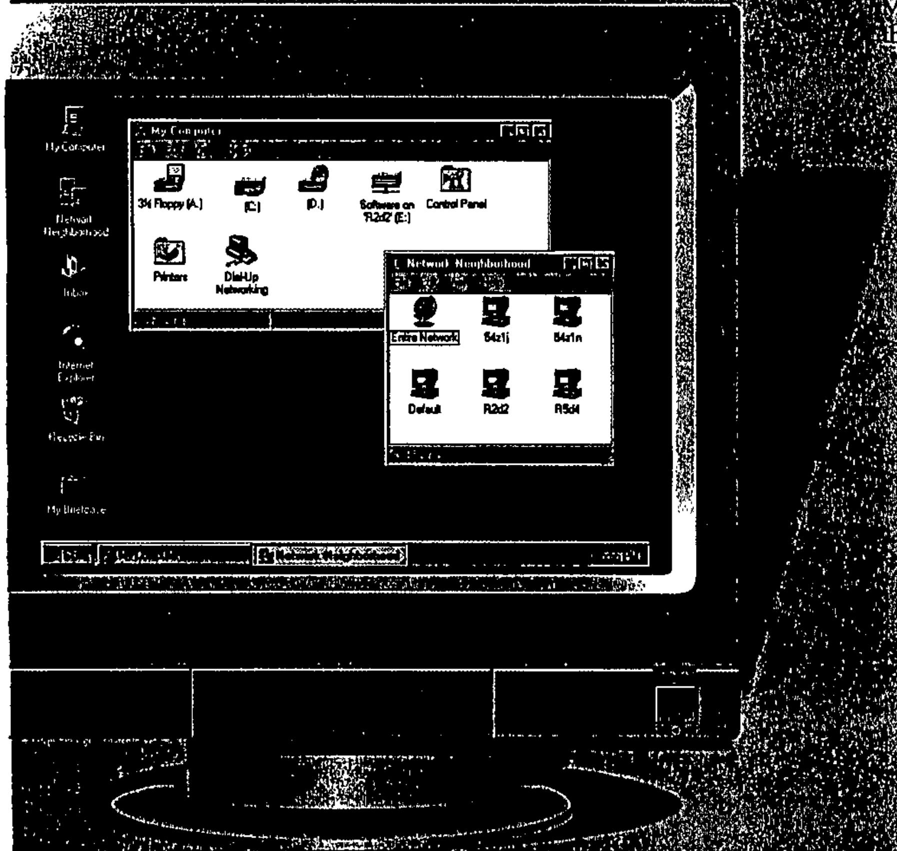
- **To disconnect a mapped network drive, right-click the drive, then choose Disconnect from the menu that appears.**
- **When you map a network drive, it behaves the same as if it were physically connected to your computer. For example, you can explore it just as you would any drive, copy and move files and folders it contains, or create a desktop shortcut to the drive.**

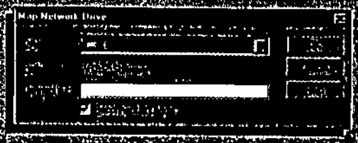
2 Next, double-click the computer icon containing the folder you need. A window containing all resources on that computer pops open.





3 Right-click the folder you want to map, then choose Map Network Drive from the menu that appears.



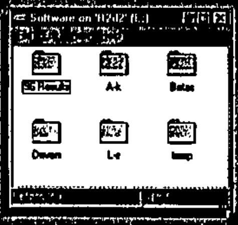


4 The Map Network Drive dialog box opens. Windows NT automatically selects the first available drive letter (in this case E:), though you can choose your own from the drop-down menu. (See the Tip Sheet for information on the Connect As field.)

5 If you want Windows NT to connect to this drive each time you log on, click the box labeled Reconnect at Logon. One caveat: choosing this option can slow the logon process a bit. If you don't choose it, Windows NT connects to the drive the first time you access it.

6 After selecting the drive letter and deciding whether to use the Reconnect at Logon option, click OK.

7 A window pops up displaying the contents of the folder you just mapped.



CHAPTER 14

Housekeeping and Gathering Information

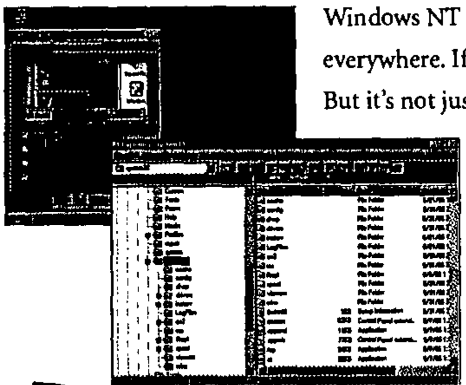
One of the strengths of Windows NT 4 is its ability to provide you with detailed information about all aspects of the operating system and the computer it's running on. It also allows you to manipulate applications and control the way they behave and perform. If you have administrator rights (which you probably do if you're running Windows NT 4 on a stand-alone machine), you can even create new user accounts and control what areas on the network each new user has access to.

In this chapter, you'll learn how to perform routine but important system tasks such as using the Windows NT 4 Task Manager, using the Windows NT 4 Diagnostics utility to tap into a treasure trove of useful information about your computer, adding a new user, and changing the performance of applications.

How to Use Microsoft Windows NT 4 Workstation

Finally, the book that lets *you* harness the power of Windows NT!

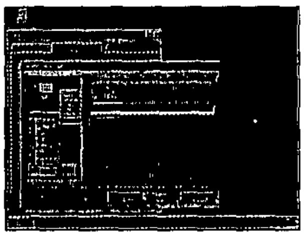
JACQUELYN
GAVRON
AND
JOSEPH
MORAN



Windows NT is fast becoming the operating system of choice in offices everywhere. If you haven't used it yet at work, chances are you will soon. But it's not just for Corporate America anymore. Now you can turn your own computer at home or work into a Windows NT workstation, and authors Jacquelyn Gavron and Joseph Moran will show you how.

In the highly acclaimed Ziff-Davis tradition, you'll be guided step-by-illustrated-step through a tour of Windows NT 4. You'll start by learning the basics of Windows NT 4 desktop features and navigation. In no time at all you'll be adding hardware seamlessly to your system and taming unmanageable files. You'll also:

- **Learn to network your machine**
- **Surf the Internet from your desktop**
- **Access your workstation from afar**



Jacquelyn Gavron is the executive editor at *Windows Sources.*
Joseph Moran is a senior analyst at *Windows Sources.*

http://www.mcp.com/zdpress/

7 2

ISBN 1-56276-445-4
52999>
9 781562 764456

Part of the **BEST-SELLING** HOW IT WORKS series

Ziff-Davis Press is an imprint of Que New Technologies, a division of Macmillan Computer Publishing USA. Macmillan also includes Sams Publishing, New Riders, Hayden Books, BradyGAMES, Sams.net, QUE Education and Training, and a software division, QUE Software. Macmillan Publishing USA is a division of Simon & Schuster, the publishing operation of Viacom.

ZIFF-DAVIS ZD PRESS
An imprint of Que

U.S.	$29.99
CANADA	$42.95
U.K.	£26.95 Net
ISBN	1-56276-445-4

BOOKSHELF CATEGORY
Operating Systems

[redacted]