PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) **International Patent Classification** 6 : **G06F 17/30** | **A1**

(11) **International Publication Number:** **WO 99/10822**

(43) **International Publication Date:** 4 March 1999 (04.03.99)

(21) **International Application Number:** PCT/US98/17444

(22) **International Filing Date:** 21 August 1998 (21.08.98)

(30) **Priority Data:**
60/056,928 22 August 1997 (22.08.97) US

(71) **Applicant** *(for all designated States except US)*: VEON, INC. [US/US]; Suite 400, 577 Howard Street, San Francisco, CA 94105 (US).

(71)(72) **Applicants and Inventors:** EFRAT, Eliahu [IL/IL]; Romano Street 36, 69018 Tel Aviv (IL). PELEG, Avner [IL/IL]; Ussishkin Street 98, 47204 Ramat–Hasharon (IL). HERMUSH, Yossi, A. [IL/IL]; Shderot Hatzionut 8, 62157 Tel–Aviv (IL). BORENSTEIN, Elhanan, A. [IL/IL]; Malachi Street 10, 63114 Tel Aviv (IL). KERET, Rottem [IL/IL]; Shchenia 120, 21042 Shchenia (IL). VERED, Eran [IL/IL]; 38930 Givat Haim Meuhad (IL). BERENSON, Adi [IL/IL]; Mendely Street 10, 43375 Raanana (IL). LAZAR, Amir [IL/IL]; Beeri Street 17, 69080 Tel Aviv (IL). ROSEN, Uri [IL/IL]; Yehuda Hamacabie Street 40, 62300 Tel Aviv (IL). PELEG, Ehud [IL/IL]; Apartment #8, Bar–Kochva Street 17, 63427 Tel Aviv (IL). WELLER, Schmuel [IL/IL]; Shalom Ash Street 21, Tel–Aviv (IL).

(74) **Agent:** HOLLOWAY, Sheryl, S.; Schwegman, Lundberg, Woessner & Kluth, P.O. Box 2938, Minneapolis, MN 55402 (US).

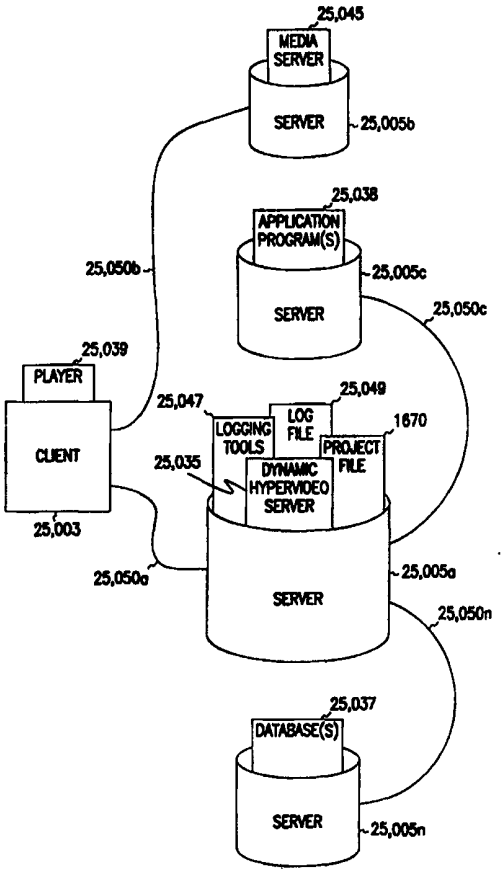
(81) **Designated States:** AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) **Title:** STREAMING HYPERVIDEO AND DYNAMIC HYPERVIDEO

(57) **Abstract**

A data stream comprises a hypervideo data stream including hypervideo data associated with an instance in time in a video. In another embodiment, the hypervideo data stream further comprises target data associated with the instance in time. A method of creating a multimedia stream comprises authoring a hypervideo, including a video. Hypervideo parameters are exported into a hypervideo data stream of a media file. In another embodiment, a data stream of the video is exported into the media file. In one embodiment, a method for authoring a hypervideo that is dynamic, comprises authoring a hypervideo, and defining a hypervideo parameter that may vary during the hypervideo performance. In another embodiment, an event handler is defined that issues a command to vary the parameter upon receiving an event message from the hypervideo. In one embodiment, a method of performing a hypervideo that is dynamic, comprises performing a hypervideo, and varying a parameter of the hypervideo during the performance of the hypervideo. In another embodiment, the parameter is varied upon the occurrence of an event in the hypervideo. In one embodiment, a hypervideo system comprises a dynamic hypervideo server. A player is coupled to the dynamic hypervideo server. A media server is coupled to the player.



FOR THE PURPOSES OF INFORMATION ONLY

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

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

Streaming Hypervideo and Dynamic Hypervideo

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyrights whatsoever.

Field of the Invention

The present invention relates generally to electronic technology, and more specifically to interactive multimedia technology.

Background of the Invention

Computers are capable of communicating information to humans in many formats, including text, graphics, sound and video. A multimedia presentation on a computer combines such formats to present information more coherently so that it is better perceived by humans.

Information on computers can be linked. For example, using "hypertext," the existence of additional and related information that is associated with a selected portion of text may be indicated by highlighting or underlining the selected text. The information associated with such selected text may be accessed, for example, utilizing a pointer device, such as a mouse. A mouse button may be actuated when a pointer is positioned on the highlighted text. After selecting the highlighted text in such a manner, the user is presented with additional relevant information.

However, text is not always the most desirable means of conveying information to humans. Some information is best illustrated with video. However, unlike text, there has not been an effective means for implementing links in a video to access information.

Summary of the Invention

The present invention solves the above-mentioned problems in the art and other problems which will be understood by those skilled in the art upon reading and understanding the present specification. The present invention is a system and method for streaming hypervideo data and for implementing dynamic hypervideos.

In one embodiment, a data stream comprises a hypervideo data stream including hypervideo data associated with an instance in time in a video. In another embodiment, the hypervideo data stream further comprises target data associated with the instance in time. In yet another embodiment, the hypervideo data stream further comprises geometry data of a hotspot associated with the instance in time. In yet a further embodiment, the hypervideo data stream further comprises shape data of a hotspot associated with the instance of time. In another embodiment, the instance in time corresponds to a frame of the video.

In a further embodiment, a multimedia stream comprises a hypervideo data stream that includes a header, including static hypervideo data. A body, coupled to the header, includes dynamic hypervideo data. The dynamic hypervideo data includes time data that is associated with an instance of time in a video with which the dynamic hypervideo data and the static hypervideo data are associated.

In another embodiment, the static hypervideo data comprises target data. In yet a further embodiment, the dynamic hypervideo data further comprises geometry data of a hotspot. In another embodiment, the dynamic hypervideo data further comprises shape data of a hotspot. In yet a further embodiment, a video stream includes a video frame, wherein the instance in time corresponds to the video frame. In another embodiment, the multimedia stream further comprises an audio stream.

In one embodiment, a method of creating a multimedia stream comprises authoring a hypervideo, including a video. Hypervideo parameters are exported into a hypervideo data stream of a media file. In another embodiment, a data stream of the video is exported into the media file.

In one embodiment, a method for authoring a hypervideo that is dynamic, comprises authoring a hypervideo, and defining a hypervideo parameter that may vary during the hypervideo performance.

In another embodiment, an event handler is defined that issues a command to vary the parameter upon receiving an event message from the hypervideo. In yet another embodiment, an event handler is defined by a logical condition including a variable defined by a query to a database. In a further embodiment, the hypervideo is authored by defining a hotspot and linking the hotspot to a target.

In one embodiment, a method of performing a hypervideo that is dynamic, comprises performing a hypervideo, and varying a parameter of the hypervideo during the performance of the hypervideo. In another embodiment, the parameter is varied upon the occurrence of an event in the hypervideo. In a further embodiment, the parameter is varied upon an event handler receiving a message associated with the event, and issuing a command to vary the parameter. In yet another embodiment, the parameter is varied upon the event handler evaluating a logical condition including a variable defined by a query to a database. In yet a further embodiment, the parameter identifies a target.

In one embodiment, a hypervideo system comprises a dynamic hypervideo server. A player is coupled to the dynamic hypervideo server. A media server is coupled to the player. In another embodiment, the hypervideo system further comprises a database. In a further embodiment, the hypervideo system further comprises an application program. In another embodiment, the application program is an electronic commerce server. In a further embodiment, the application program is an advertising server. In another embodiment, the application program is a training server. In a further embodiment, the media server is a video server. In another embodiment, the hypervideo system further comprises a network coupling the player to the dynamic hypervideo server and the media server.

It is a benefit of the present invention that it facilitates streaming hypervideo data over networks. It is a further benefit of the present invention that it permits the parameters of a hypervideo to be varied during the hypervideo

performance. Further features and advantages of the present invention, as well as the structure and operation of various embodiments of the present invention, are described in detail below with reference to the accompanying drawings.

Brief Description of the Drawings

Figure 1A illustrates an exemplary hypervideo environment, including a hypervideo authoring tool;

Figure 1B illustrates an exemplary hypervideo run-time module;

Figure 1C illustrates an exemplary block diagram of one embodiment of a computer system;

Figure 1D exemplifies a computer implemented as an integrated circuit;

Figure 1E illustrates an exemplary television;

Figure 2 illustrates an exemplary Import Media File Dialog;

Figure 3 illustrates an exemplary Import From Project Dialog;

Figure 4 illustrates an exemplary Media Warehouse Window Icon;

Figure 5 illustrates an exemplary Details Page;

Figure 6 illustrates an exemplary Preview Page;

Figure 7A illustrates an exemplary Targets Page;

Figure 7B illustrates an exemplary Hotspot Page;

Figure 7C illustrates an exemplary General Properties Page;

Figure 7D illustrates an exemplary Time Line;

Figure 7E illustrates another exemplary Details Page;

Figure 7F illustrates an exemplary Frame Window;

Figure 7G illustrates an exemplary Place Properties Page;

Figure 8A illustrates an exemplary Workshop Window;

Figure 8B illustrates an exemplary Preview Window;

Figure 9A illustrates an exemplary Tools Window;

Figure 9B illustrates an exemplary Tracking Property Sheet;

Figure 10 illustrates an exemplary graphical user interface (GUI) for a hypervideo authoring tool;

Figure 11 illustrates an exemplary hypervideo story board;

Figure 12 illustrates an exemplary call-back object;

Figure 13 illustrates an exemplary object tree;

Figure 14 illustrates an exemplary hypervideo data stream;

Figure 15 illustrates an exemplary GUI of a modified version of an authoring tool for Netshow (TM);

Figure 16 illustrates an exemplary Hotspot Properties Sheet;

Figure 17 illustrates an exemplary Media Properties Sheet;

Figure 18 illustrates an exemplary Properties Page;

Figure 19 illustrates an exemplary Active X (TM) control;

Figure 20 illustrates an exemplary GUI of a modified version of an authoring tool for RealVideo (TM);

Figure 21 illustrates an exemplary Hotspot Properties Sheet;

Figure 22 illustrates an exemplary Export Dialog;

Figure 23 illustrates an exemplary Export Dialog for automatic translation;

Figure 24A illustrates an exemplary Encoding Properties Sheet;

Figure 24B illustrates an exemplary Encoding Properties Page;

Figure 25 illustrates an exemplary hypervideo system;

Figure 26 illustrates an exemplary authoring tool GUI;

Figure 27 illustrates an exemplary Data Source page of Query dialog;

Figure 28 illustrates an exemplary Choose Columns Page;

Figure 29 illustrates an exemplary Filter Data Page;

Figure 30 illustrates an exemplary SQL Page;

Figure 31 illustrates an exemplary Targets Page;

Figure 32 illustrates an exemplary Hotframes toolbar;

Figure 33 illustrates an exemplary Hotframes Property Sheet;

Figure 34 illustrates an exemplary Properties Sheet;

Figure 35 illustrates an exemplary Handlers Notebook Window;

Figure 36 illustrates an exemplary Script Wizard;

Figure 37 illustrates an exemplary Menu;

Figure 38 illustrates an exemplary Default Settings Sheet;

Figure 39 illustrates an exemplary Logging Page;

Figure 40 illustrates an exemplary flow diagram illustrating the procedure for creating a hypervideo;

Figure 41 illustrates exemplary logical class relationships associated with the Media Warehouse Window;

Figure 42 illustrates exemplary classes associated with class EClipDoc;

Figure 43 illustrates exemplary classes associated with class EClipView;

Figure 44 illustrates exemplary classes associated with EClipFrameWnd.

Figure 45 illustrates an exemplary run-time tree class diagram;

Figure 46 illustrates an exemplary video object class diagram;

Figure 47 illustrates an exemplary sound object class diagram; and

Figure 48 illustrates exemplary classes associated with a hypervideo.

Detailed Description of the Embodiments

In the following detailed description of the preferred embodiments, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration specific preferred embodiments in which the invention may be practiced. These embodiments are described in sufficient detail to enable persons skilled in the art to practice the invention, and it is to be understood that other embodiments may be utilized and that logical, mechanical and electrical changes may be made without departing from the scope of the present invention. The following detailed description is, therefore, not to be taken in a limiting sense, and the scope of the present invention is defined only by the appended claims.

Table of Contents

1.0 Introduction 8

2.0 Software Implementation 9

2.1 Authoring Tool 10

2.11 The Menu Bar 12

2.12 Main Tool Bar 17

2.13 Media Warehouse 17

2.14 The Workshop 24

2.141 Hotspot Definition 25

2.142 Hotspot Tracking 30

2.143 Targets 33

2.15 Project View 47

2.2 Run-Time Module 47

2.21 Run-Time Module Commands 49

2.22 Run-Time Module Design 49

2.3 Streaming Hypervideo 61

2.4 Dynamic Hypervideo 108

2.5 Object Oriented Implementation of Hypervideo System 149

3.0 Conclusion 207

1.0 Introduction

The present invention is a method and apparatus for linking information to and accessing information from a video. The video may be hypervideo. Hypervideo is video with one or more regions of interest, where each region of interest may be linked to one or more targets. Hypervideo permits a user to interact with video. A user can create hypervideo which is non-linear. As a result, for example, the user can navigate the hypervideo from a base target, such as, but not limited to, a video, to other target(s), such as, but not limited to, HTML files or other videos. The other targets may be executed, or activated, simultaneously, sequentially, or a combination thereof. The parent target can be halted, for example when paused or closed, or keep playing.

The present invention comprises a method and apparatus for creating and playing hypervideo. The present invention may be implemented with computer programs. The program that creates hypervideo is known as an authoring tool, or an editor. The program that enables a computer to display hypervideo is known as the run-time module.

The authoring tool may be used to define a hotspot, or an object, in a region of interest in one or more frames of a video, and then track the hotspot in later frames, for example. The hotspot may also be defined in a picture, including a bitmap. Any subsequent discussion of bitmaps may also be applicable to other picture formats, which are subsequently described. Hotspots can be alternatively tracked manually and automatically. With the authoring tool, the hotspot may be linked to a target, for example, but not limited to, text, audio, or a second video. Targets are further described and exemplified below. Thus, for example, when playing the hypervideo with the run-time module, a user can place a pointer over the hotspot in the video with a mouse. When the pointer is placed over the hotspot, the cursor may change. Then, by actuating a mouse button, the user may launch, or execute, the target. When the mouse button is actuated when the pointer is over the hotspot, the cursor may change again. Multiple targets can be linked to a hotspot. Because of its ease of use, hypervideo can be used for a wide variety of applications, including, but not limited to, interactive television, games, tourism and home shopping.

2.0 Software Implementation

Exemplary structures of computer programs for authoring and playing hypervideo are respectively shown in Figures 1A and 1B. The computer programs comprise a hypervideo environment 1000 including an authoring tool 1001 and a run-time module 1101. The computer programs may be implemented with object-oriented software, as exemplified below. The computer programs can be executed on a computer, for example, using the Windows 95 operating system by Microsoft Corporation (Redmond, Washington). Thus, the programs, specifically the authoring tool 1001, may have the look and feel of Windows 95. However, the present invention may be implemented, for example, in other systems, such as, but not limited to, televisions, described below. The present invention may also be implemented with other operating systems.

Figure 1C illustrates an exemplary computer system 1605 in which one or more programs or sub-programs of the hypervideo environment 1000 may reside and be executed. The computer system 1605 may be any processing system for executing one or more programs or sub-programs of the hypervideo environment 1000, including, but not limited to, personal computers and interactive televisions.

The computer 1605 may include a central processing unit 1610 and memory 1620. The processing unit 1610 may be, but is not limited to, a Pentium microprocessor by Intel Corporation (Santa Clara, California). The memory 1620 can be random access memory, disk storage, CD-ROM storage, digital video, or versatile, disk (DVD) 1159, another type of memory or combinations thereof. The memory may be a video source. Within the memory 1620, the computer system 1605 has access to its operating system 1630 and user software 1640. The user software 1640 can include the authoring tool 1001, the run-time module 1101, stand-alone modules 1107, multimedia control interface (MCI) driver 1109, software development kit (SDK) 1111, plug-ins 1103, and the hypervideo project file 1670.

In one embodiment, the computer 1605, or portions thereof, may be implemented in an integrated circuit. For example, as illustrated in Figure 1D, the integrated circuit 1695 can be implemented in a DVD apparatus 1625 including a

reader 1627 for reading information from a DVD, operatively coupled to the integrated circuit 1695. In another embodiment, the computer 1605, or portions thereof, may be implemented in a television 1685, as illustrated in Figure 1E, for receiving hypervideo television signals. The television 1685 also includes a receiver 1687 which may be coupled to the computer 1605 or portions thereof. The computer may permit displaying a hypervideo, for example, on the television.

The hypervideo project file 1670 is a database used by both the authoring tool 1001 and the run-time module 1101. The extension of hypervideo database files may be *.OBV. The project file 1670 may be encrypted creating encryption keys using functions, such as srand and rand. The authoring tool 1001 and run-time module 1101 may include components that detect the authoring tool 1001 version used to create a project file 1670 and thus can read the project file 1670. Thus, the authoring tool 1001 and run-time module 1101 may be upgraded without project files created by older versions of the authoring tool becoming obsolete or unusable. The authoring tool 1001 and the run-time module 1101 will now be successively described.

2.1 Authoring Tool

The authoring tool 1001 may be used to define hotspots in media, such as video, and link 1010 the hotspots to targets. Hotspot definition, for example, may be performed substantially in real time. The authoring tool 1001 comprises three interconnected sub-programs, the media warehouse 1003, the workshop 1005, and the project view 1007. Each sub-program may be represented by a window on the display of a computer. Furthermore, apparatuses and methods described below for the authoring tool 1001 may also be used in the run-time module 1101, or vice versa. The authoring tool 1001 may plug into other editing tools, such as Adobe Premier by Adobe Systems Incorporated (San Jose, California).

The authoring tool 1001 may include a registry 1006. The registry 1006 permits each user to save their own authoring tool settings, including the placement of windows. The registry 1006, for example, may be implemented with the registry system of Windows 95 or Windows NT by Microsoft.

A user may utilize the authoring tool 1001 to perform the following tasks. The user may import media files 1680 into the media warehouse 1003. Each imported media file 1680 may result in a new media element 1690 being created in the media warehouse 1003. Each media element 1690 may include a reference, which may be a pointer, to media files 1680 which could be video, bitmap, sound, or text. The pointer may also be directed to non-media, such as a uniform resource locator (URL) or an executable file, which are considered to be media with respect to media files 1680 and media elements 1690 in this document. One or more media files 1680, generally video and bitmap files, may be selected to form a base target. Then, these video and bitmap files may be placed in the workshop 1005 so that hotspots may be defined within them. After the hotspots are defined, each hotspot may be linked 1010 to one or more other targets, which are media elements 1690. The base target and other targets form a hypervideo story board 1116. The aforementioned techniques may be described as nonlinear authoring 1009.

The hypervideo data created by manipulating the different sub-programs of the authoring tool is stored in a project file 1670. The hypervideo data may include, for example, but is not limited to, data pertaining to media elements 1690, hotspots, targets, and cursors. The project file 1670 can be exported to video servers, for example, that provide video on a network 1650, such as the Internet or an Intranet. Alternatively, it is envisioned that hypervideo data may be exported by the authoring tool 1001, for example, into a media file 1680 or into a data stream 1660 that is parallel with streams of video and audio information. For example, the data may be stored in enhanced video files. Enhanced video files include hypervideo information in the video file. One form of an enhanced video file, such as an AVI file, may only include hotspot information. Enhanced video files may also be recognized in the hypervideo environment 1000.

The authoring tool 1005 includes a graphical user interface (GUI) to facilitate authoring hypervideos. However, a scripting language 1004 can also be used to author hypervideos. The GUI includes the following components:

A **Menu bar**;

A **Media Warehouse** window – which contains the media elements 1690 including the references to the media files 1680 in a hypervideo project;

A **Workshop** window – in which the user defines the hotspots, and links the hotspots to targets. From the Workshop window the user can also open the Preview, or Cinema, window to review the hypervideo project at any time.

A **Tools** window – which holds tools with which the user may define the hotspots and link 1010 the hotspots to targets;

A **Preview** window – with which the user may preview the hypervideo project; and

A **Project View** window – in which the user can view and may also edit the logical flow of a hypervideo project.

The GUI will be subsequently described in further detail.

2.11 The Menu Bar

The authoring tool may include a menu bar to manipulate the Media Warehouse, Workshop and Project View windows. The menu bar may include the following commands:

Command	**Description**
File	Handles file commands. The file entity is either a hypervideo project file 1670 or a media file 1680.
New	Creates a new project file 1670.
Open...	Opens an existing project file 1670. May also preview the corresponding hypervideo in a project view window.
Save	Saves the project file 1670 with its current name.
Save As...	Saves the project file 1670 with a new name.
Revert	Retrieves the last saved version of the project file 1670. This operation closes the current file and opens its previously saved version.
Project Settings	
Base Target	Subsequently described.
Default Cursors	Subsequently described.
Coordinate Resolution	Subsequently described.

Command	Description
Import Media File...	Imports a media file 1680 or folder. This option lets the user select a directory and display all media files 1680 in that directory. The user can select one, several or all media files 1680 in the specified directory. See Figure 2. The user may drag and drop files or folders from an open import media file dialog into the Media Warehouse window.
Import From Project...	Imports one or several media elements 1690 from a selected, existing project file 1670. This option lets the user utilize previously defined hotspots in the media elements 1690. The media elements 1690 are associated with the selected project. An imported media element 1690 includes all defined hotspots within the media element 1690. This command saves the user time when defining hotspots. See Figure 2.
Import Cursor	Imports cursors to a hypervideo project. May be alternatively accessed by the browse command, described later. All cursors that are imported to the hypervideo project can be displayed in a cursor window. The cursor window can be a simple list view which enables the user to import new cursors into the project. When selecting a cursor for a hotspot, the user can drag a cursor from the cursor window to the Hotspot Properties Sheet. Animated cursors may be shown in their animated form in the cursor window. The cursor data need not be stored in the project file 1670.
Preview	Runs the hypervideo project from its beginning. Also may be initiated by the preview button in the Workshop window.
Print	Prints reports of the hypervideo project. The following reports can be printed: Project Scheme – Illustrates the story board 1116 created in the authoring tool 1001. Project Probabilities 1096 – Calculates and prints a report of the probability of reaching each branch in the hypervideo. Project Listing – Prints a list of all hotspots in the form of an identifier (ID) (i.e. text)
Print Setup...	Sets up a printer. Standard dialog for setting up the print environment.
Exit	Exits the program.
Edit	
Undo	Undoes the last action.

Cut		Cuts the selected item and places it in the clipboard. This action may be context related. Thus, when the user selects this command, the action performed by the authoring tool 1001 is related to the item that is selected (i.e., hotspot or media element 1690). The Cut command may be a multiple selection command, as described below.
Copy		Copies the selected item in the clipboard. This action is context related. This means that when this command is used, the action performed by the authoring tool 1001 is related to the item that is selected (i.e., hotspot or media element 1690). The Copy command is a multiple selection command.
Paste		Pastes the clipboard contents into the specified location. See the Cut command about context relation.
Delete	Del	Deletes the selected item whether from the Media Warehouse window or the Workshop window.
Select All		Select all items. Applicable both to the Workshop and the Media Warehouse windows.
Validate		Validates a hypervideo project by searching for dead locks, errors, and minor authoring mistakes. This operation may generate a report that can be printed and reviewed by the user. Different levels of warnings are permitted.
Command		**Description**
Media		Handles commands for manipulating the Media Warehouse window.
Partial View		View of the media elements 1690 without their details. Just the name of the file and a thumbnail are displayed.
Detailed View		View of the media elements 1690 with all details and a header for each of the categories. The user can actuate a header to sort the media elements 1690 according to the selected category.
Sort by		Sort the media elements 1690 in the Media Warehouse window by a specified category.
View only		View only a certain type of media file 1680 (e.g., video, sound, bitmap, etc.).
Workshop		Manipulates the Workshop window.
Preview Mode		Enters preview mode in which the hypervideo project plays and permits navigation through the hypervideo with hotspots.

Play / Pause	Plays and Pauses the video displayed in the Workshop window. These commands appear depending on the state of the Workshop window. If Workshop window is playing a video, the command shows a Pause button. However, if the video is paused, the Workshop window shows a Play button.
In	Marks a currently displayed frame as default IN for the media, such as video.
Out	Marks the currently displayed frame as default OUT for the media, such as video.
Home	Goes to the IN frame of the video in the Workshop window.
Define New Hotspot	Defines a hotspot of the selected shape (e.g., Elliptic, Rectangular, Triangular, or Polygon).
Set Target	Enters the Set Target mode, as described below.
Options	Authoring Tool options.
Command	**Description**
Settings	May pop up a properties sheet with settings for all windows and the hypervideo project. The properties sheet may be used to define run-time details, including the size and position of the Preview window, as well as assign the base targets of the project. The base targets may play when the project is started. To define base targets, the user may drag video media elements, for example, from the Media Warehouse 1003 to the Targets, or Links, page of the properties sheet. Also, the properties sheet may permit controlling automatic tracking, hotspot colors and patterns (typically crosshatched), maximum number of polygon vertices and other workshop parameters. A hotspot may have three states: Selected, Deselected and Interpolated. The user can choose colors and patterns for each state. The maximum number of polygon vertices limits the number of vertices that may be used to define a hotspot in a freehand mode.
Window	Controls the windows of the authoring tool 1001.
Media	Makes the Media Warehouse window appear.
Workshop	Makes the Workshop window appear.
Tools	Makes the Tools window appear.
Close	Closes the selected window.
Close All	Closes all windows.

Command Dialogs

Selection of specific menu bar commands may launch dialogs for importing media files. Specifically, dialogs are launched when placing media elements 1690 into the media warehouse 1003 with the import media file and import from project commands. These dialogs are described below.

Import Media File

In order to start building a hypervideo project, the user may import media files 1680 into the media warehouse 1003. Figure 2 is an illustration of an exemplary Import Media File dialog 2001. The Import Media File dialog 2001 is launched upon the selection of the import media file command. The Import Media File dialog 2001 permits the user to view media files 1680 and import corresponding media elements 1690 into the media warehouse 1003. The user can import media elements 1690 in the following ways:

1. Import a media element 1690 corresponding to a media file into the media warehouse 1003.
2. Import media elements 1690 corresponding to media files 1680 in a directory or file folder into the media warehouse 1003.

This dialog 2001 may use a multiple selection list. The Import Media File dialog 2001 may permit several files or an entire directory to be selected.

When actuating a Preview button 2003, the dialog may show a preview of the selected file, with the ability to execute, for example, a video, audio or bitmap media. Media files 1680 and folders may also be dragged and dropped, for example, from Explorer by Microsoft and other file managers.

Import From Project

Figure 3 is an illustration of an exemplary Import From Project dialog 3001. The Import From Project dialog 3001 permits the user to view and import media elements 1690 from a hypervideo project file 1670 into the media warehouse 1003. A preview window may be opened to display both the hypervideo project and corresponding media files 1680 by actuating the Preview button. In the preview window, hotspots may be shown. The Import From

Project dialog 3001 includes features of the Import Media File dialog 2001 described above.

2.12 Main Tool Bar

The hypervideo authoring tool 1001 may have a main tool bar. The main tool bar may float and can be closed. The main tool bar functions may include: New, Open, Save, Import Media File, Print, Preview, Copy, Cut, Paste, Media view formats, Options | Settings, Window Tiles, and Help.

2.13 Media Warehouse

The Media Warehouse 1003 may contain media elements 1690 that are used to form a hypervideo project. Each media element 1690 may include a reference to a media file 1680 which could be video, bitmap, sound, or text. The pointer may also be directed to non-media, such as a uniform resource locator (URL), a hypertext markup language (HTML) file, or an executable file, as described above. For example, the URL may point to HTML file. The video and bitmap media elements may include hotspots that are linked 1010 to targets, such as other media files 1680.

The Media Warehouse Window

The Media Warehouse 1003 may be represented by a window on a display on a computer or television, for example. The Media Warehouse window displays a thumbnail of the contents of a media file 1680 pointed to by a media element 1690. The Media Warehouse window may permit the user to view the media elements 1690 in one of several views, including:

1. Symbolic view – in which, for example, only thumbnail illustrations of the contents of media files 1680, referenced by corresponding media elements 1690, are displayed; and
2. Detailed view – in which detailed media element 1690 information is displayed.

Each media element 1690 has its own identification (ID) section in the Media Warehouse window. As shown in Figure 4, the media element's ID section, or icon, 4001, may consist of:

- A thumbnail illustration 4007 of the contents of a media file 1680 pointed to by a media element 1690. The thumbnail illustration 4007 is a picture in the case of bitmap or video media files, and a waveform in the case of sound media files. For video media, the picture may show one frame, presumably the first frame, of the video media file.
- An icon specifying the data type of the media file 4005 for any view that does not require a specification of the content of the media file 1680, but rather the data type and name of the media element 1690.
- Textual data that describes the media file 1680.

In the Media Warehouse window, if a media element 1690 includes at least one target, the right side of the media element icon 4001 may be decorated by a token 4003, or link button. This token 4003 indicates that the media element 1690 is linked 1010 to a target, such as a media file 1680. The token 4003 may also be a button. Upon actuating the mouse button when a pointer is over the token 4003, the user may be presented with a pop-up window that shows all targets that are linked 1010 to the selected media element 1690. The user may drag the pointer over a target and actuate the mouse button. Then, the selected target may scroll up or down to the top left position in the Media Warehouse window. The pop-up window may then show any targets linked 1010 to the selected target. This method provides an easy and intuitive method to navigate through the files in the Media Warehouse window. The linked target pop-up window may be closed by actuating the mouse button when the pointer is over the token 4003 of the selected media element. Figure 4 shows an exemplary media element icon 4001 for a video in the Media Warehouse window.

Alternatively, the token 4003 may be implemented as an icon, and not a button. As a result, the Media Warehouse window can be implemented as a common list view control.

The authoring tool 1001 can show targets linked 1010 to hotspots in a manner that is more easily understood by a user. Hotspots, rather than links

1010, can be viewed by manipulating the media element icon 4001. For example, when the pointer is positioned over a media element icon 4001 and the right mouse button is actuated, a menu of hotspots, rather than linked targets in the selected media element, is displayed. Then, the linked targets of each hotspot may be displayed. These techniques will subsequently be described.

The authoring tool 1001 supports the 8, 16, 24, and 32 bit color modes. Thus, thumbnails and previews may be displayed in the color mode of the corresponding media file 1680, or alternatively, in the 24 bit color mode.

Relative Path

The authoring tool 1001 may create a project file 1670 that defines the paths of media files 1680, corresponding to media elements 1690 in a hypervideo, relative to the path of the project file 1670. The common portion of the paths of the project file 1670 and media file 1680 is referred to as the static path 1163. The unique portion of the media file 1680 path is referred to as the relative path 1161. The project file 1670 and media files 1680 may be stored locally or remotely 1139. For example, the project file 1670 and media files 1680 may be moved to a different memory unit, such as disk drive, computer, or computer network, while keeping the relative path 1163 the same, then the project file 1670 does not require editing for the hypervideo to be performed. This will enable altering the drive letter or even the base path from which the hypervideo project is executed.

Tool Tips

A tool tip can be activated and display information about a media element 1690 when a pointer is positioned over a media element icon 4001 in the Media Warehouse window. The tool tip, for example, may show the full or relative path 1161 of the corresponding media file 1680. The tool tip may include other information, such as the type, and logical and physical size of the corresponding media file 1680.

Media Warehouse Window Controls

The Media Warehouse window may be implemented using a custom list view control from the Microsoft Windows 95 common controls. The list view may show the media elements 1690 in different views (i.e., with details or icons), and permits user manipulations of the lists, such as sorting and reordering. Each media element 1690 of the Media Warehouse window list corresponds to a media file 1680, such as a bitmap. The Media Warehouse window list is associated with an image list control that includes a thumbnail illustration 4007 and a description of the type of the media file 1680. The text of the Media Warehouse window list describes file parameters and are displayed to the user in a detailed view.

The Media Warehouse window can be manipulated in the following ways. Using the list view, the user can, for example:

- Resize the list to any size, and create a column like list or a scattered list.
- Change the order of the media elements 1690 by dragging and dropping media element icons 4001 corresponding to the media elements 1690, inside the list.
- In the detailed view, sort media elements 1690 by category, including media file 1680 type and size. Sorting may be performed by actuating the mouse button when the pointer is over the desired list view header.
- Actuate the alternate mouse button to obtain a list of all targets linked 1010 to, or all hotspots in, a specific media file 1680. By selecting a specific target, for example, the user can access the corresponding media element 1690 in the list.
- Double click the mouse button when the pointer is over a media element 1690 to preview the corresponding media file 1680. This option may also be activated with the alternate mouse button.
- Drag and drop media elements 1690 into other windows of the authoring tool 1001, such as the Workshop window. For example, when a media element icon 4001 is dropped into the Workshop window, the respective media file 1680 is opened and, for example for a video, its first frame is displayed.

The lists of the Media Warehouse window may support full multiselect operation for the delete and move functions.

Alternate Mouse Button

The alternate mouse button can be actuated to perform the following functions, depending on the media file 1680 type of the media element 1690 that is selected.

Preview – open a viewer for that media file 1680 type. Videos that are linked 1010 may not display any hypervideo capabilities in the viewer.

Send to Workshop – The project file 1670 will be opened in the Workshop window (see subsequent description).

Links ▸ – May open a list of targets (e.g., media elements 1690) that are linked 1010 to the selected media element 1690. Choosing one of the linked targets will move the list to the selected target. Alternatively, a list of hotspots, not necessarily linked 1010 to targets, in the selected media element 1690 is displayed. Selecting one hotspot will display a list of the targets linked 1010 to the hotspot. The latter option enables fast and intuitive navigation through component media elements 1690 of a hypervideo project.

Insert...– Imports a new media file 1680 and creates a corresponding media element 1690. The corresponding media element 1690 is placed before the selected media element 1690 in the list view window. The new media file 1680 can be previewed when performing the Insert function.

Delete – Deletes the selected media element 1690 from the Media Warehouse window.

Properties...– Permits viewing and editing of media properties of the selected media element 1690.

Editing Options

The media list of the Media Warehouse window may support the following editing commands:

- Undo – Upon deleting or moving a media element 1690 in the list.
- Delete – Permits multiselect deleting.
- Cut
- Copy
- Paste
- Select All - Selects all media elements 1690 currently in the list.

Viewing Options

The user can enable the Media Warehouse window to display only media elements 1690 corresponding to certain types of media files 1680, such as only video or sound files. Furthermore, the media elements 1690 in the Media Warehouse window can be manually arranged by the user. Typically, though, the media elements 1690 are automatically arranged.

Media Properties Sheet

Properties for each media element 1690 may be shown in a Media Properties Sheet. The Media Properties Sheet may be a Win32 properties sheet with tabs that represent each page in the properties sheet. The Media Properties Sheet may include a Details, or General, page. The Details page includes general file information relevant to the specified media type. Figure 5 illustrates an exemplary Details page 5001 of the Media Properties Sheet 5003.

The Details page 5001 may include the file name and path of the media file 1680. The media file 1680 name and path may be too long to be viewed on the Details page 5001. Therefore, only the media file 1680 name may be displayed in the Details page 5001. The path may be displayed in a tool tip when the user moves a pointer over the file name. The Details page 5001 also may include default cursors of the media element 1690.

The Details page 5001 may also incorporate general file data such as size 5005 and location 5007, together with creation and modification dates 5009, 5011. One section of the Details page 5001 may include data pertaining to the

type 5019 of media file 1680 to which the corresponding media element 1690 and Media Properties Sheet 5003 are associated. Relevant data for different media file types 5019 is described below:

- Video – the pertinent data may include the length in seconds 5013, and the audio and video formats 5015, 5017. Another parameter is the manual tracking speed for the video. Each video may have an optimal manual tracking speed. The user may control the manual tracking speed with a slider.
- Bitmap – the pertinent data may include the pixel size of the bitmap, number of colors, and compression type if applicable.
- Audio – the pertinent data may include the audio format and length in seconds.

The second page of the Media Properties Sheet 5003 may be a Preview page. Figure 6 illustrates an exemplary Preview page 6001 for a video. Videos may, for example, be displayed in either 320 by 240 or 160 by 120 pixel areas. Bitmaps may be viewed in a fixed pixel area of 160 by 120. Audio emissions and video frame display may be controlled with a slider 6007 to manipulate the audio and video file advances, and also possibly to display a corresponding waveform of the audio file. Text may be shown in a scrolled window.

The third page of the Media Properties Sheet 5003 may be the Targets, or Links, page. Figure 7A illustrates an exemplary Targets, or Links, page 7001. In the Targets page 7001, the user can view a list of targets 7003 that are linked 1010 to hotspots in the selected media element 1690. The Targets page 7001 also includes an Add Target button 7005. The user actuates the Add Target button 7005 and holds the mouse button down just as if the user were clicking the video hotspot in the Set Target mode. The cursor changes to the Set Target cursor, and the process of setting a target begins as will be subsequently described. Other methods for linking 1010 targets to hotspots may be used.

The Media Properties Sheet 5003 may also have a Hotspot page 7011, exemplified in Figure 7B, that illustrates the hotspots in a selected media element 1690. The Hotspot page 7011 may display a list of hotspot locations in thumbnail illustrations 4007. For a video, the thumbnail illustration 4007 may be

the first frame in which the hotspot appears. Placing the pointer on a thumbnail illustration 4007 of a selected hotspot in the list and actuating, for example, by double clicking, the mouse button may cause a corresponding Hotspot Properties Sheet to be opened. The Hotspot Properties Sheet 7101 may include the Targets page 7001, which, in this case, may only display a list of targets 7003 linked 1010 to the selected hotspot.

The Hotspot Properties Sheet 7101 may also include a General Properties page 7103 exemplified in Figure 7C. The General Properties page may include the hotspot name 7103, type of shape 7105 defining the hotspot, media file name in which the hotspot is located 7107, the range of frames in a video media file in which the hotspot 8003 is located 7109, and cursor types. The cursor types include the cursor displayed when the pointer is over the hotspot 7111, and the cursor displayed when the mouse button is actuated when the pointer is over the hotspot 7113.

2.14 The Workshop

The Workshop 1005 may be used to define hotspots 1023 and create links 1010 between hotspots and targets. Hotspots, for example, may be created in videos and bitmaps. The project may be viewed in the Preview, or Cinema, window. The Workshop 1005 is displayed in a Workshop window that may have a toolbar window attached to it. The Workshop window changes in accordance with the tool selected from the Tools window. The Workshop window has an interface that permits selecting a specific frame in a video, and forwarding or reversing the displayed video frame. When displaying bitmaps, video frame position controls are disabled. Figure 8A illustrates an exemplary Workshop window 8001 displaying a frame 8013 of a hypervideo.

In the Workshop window 8001, the user may define and edit hotspots 8003 in video, for example, and link 1010 the hotspots 8003 to targets 7003. The user can preview the hypervideo project by toggling the Preview, or Cinema, switch 8005 to the ON position to pop up the Preview window. The Preview switch 8005 may, however, be a button.

The Workshop window 8001 may include the following features items:

Play button 8007 – plays and pauses a video. The play button, as for many player interfaces, turns into a Pause button while the video plays.

Home (or In) button – displays the IN frame (default is the first video frame).

End (or Out) button – displays the OUT frame (default is the last video frame).

Slider 6007 – permits the user to select a desired frame of a video. The frames of the video are displayed as the slider is moved.

Frame 8011 +/- buttons – respectively moves one frame 8013 forward (+) and backward (-), and is useful for pinpointing a particular frame.

Scale Information – gives the user frames/time information.

Preview switch 8005 - opens the Preview window.

The Workshop window 8001 may serve as a video viewer. With the Tools window, a user can also use the Workshop window 8001 to define hotspots 8003 and links 1010 to targets 7003. The Preview and Tools windows will now be described.

The Preview Window

The Preview window is a hypervideo player in the authoring tool 1001. In the Preview window, the user can execute targets by activating, or actuating, hotspots in the hypervideo, for example. The Preview window 8101, illustrated in Figure 8B, may be similar to the Workshop window 8001 except for the fact that it is a video viewer that may not permit editing. In this case, the IN/OUT and Preview 8005 switches are absent.

2.141 Hotspot Definition

Hotspots 8003 are defined in media, such as video or bitmaps, in the Workshop window 8001 with the methods described below.

The Tools Window

The Tools window 9001 includes one or more dockable toolbars that are used to define hotspots 8003 and link 1010 hotspots 8003 to targets 7003. Figure 9A illustrates an exemplary Tools window 9001. Typically, the Tools window 9001 may be attached to the Workshop window 8001.

The Tools window 9001 can dock on any side of the Workshop window 8001. The Tools window 9001 is a toggle toolbar. Thus, only one button (i.e., mode) can be used at any instance of time. The Pick Hotspot button 9003 may be the default selection mode of the Tools window 9001. Each selected mode changes the behavior of the Workshop window 8001. The Tools window 9001 modes will now be described:

Pick Hotspot

The Pick Hotspot mode 9003 permits the user to select and manipulate a defined hotspot 8003. In this mode, the user can:

- Select a hotspot 8003 or a group of hotspots.
- Reposition a hotspot 8003 in the current frame.
- Resize a hotspot 8003 in the current frame.
- Double click on the hotspot 8003 and view its properties.
- Press Del (Delete) and delete the hotspot 8003.

Holding down the SHIFT key while moving the Workshop window slider 6007 marks a selection of frames. Any action that is performed in the Pick Hotspot mode 9003 will be executed throughout the range of selected frames.

Define Hotspot

The Define Hotspot mode 1025 permits definition, or segmentation, of new hotspots 8003. When selecting the Define Hotspot mode 1025, all previously selected hotspots 8003 are deselected. The hotspot is then defined by drawing the desired shape 91037. The user may select a primitive shape 1039 from the Tools window 9001 shown in Figure 9A. The primitive shape 1039 may be, but is not limited to, an ellipse 9009, a triangle 9007, or a rectangle 9005. The primitive shape 1039 may be default-sized. However, a polygon 1041 may

also be selected. The user may then place the selected shape in the media in the Workshop window 8001. After the shape has been placed, the Tools window 9001 may automatically switch to the Pick Hotspot mode 9003. Then, the user can manipulate the size and position of the selected shape to define the hotspot 8003. Hence, the hotspot's geometric form may be defined.

Hotspots 8003 having the shape of a polygon 1041 can be defined, for example, in one of three ways:

(1) With a magic wand 9011, described below;

(2) With a polygon drawing tool with which the user draws lines between polygon vertices; and

(3) With freehand drawing, the user draws the hotspot 8003 in freehand form and the Workshop 1005 calculates the optimal number of vertices limited by the maximum number of vertices parameter.

Hotspots 8003 having the shape of a polygon can be re-sized using a bounding rectangle or by moving each vertex of the polygon. The user can select either mode in the Workshop window 8001.

Upon its creation, a new hotspot 8003 may be given a default name, which may be changed later. The hotspot name may be added to a list of defined hotspots 8003 in the Workshop's Tools window 9001, which may also include a Hotspot toolbar. If the user double-clicks the mouse button when the pointer is over the hotspot 8003, the Hotspot Property Sheet 7101 may be displayed. In the Hotspot Property Page, the user can assign a different name to the hotspot 8003, and the cursors that are to be used when the pointer is positioned over the hotspot 8003 or when the hotspot 8003 is activated.

Furthermore, the Hotspot toolbar allows the user to select a hotspot 8003 and display the first frame in which that hotspot 8003 appears in a video. The user can also open the Hotspot Property Sheet 7101 from the Hotspot toolbar as well as delete the currently selected hotspot 8003 and all of its occurrences.

Magic Wand

The Magic Wand 9011 is used to define a hotspot 8003 automatically 1033, rather than with shape drawing 1037. When the user points the Magic Wand 9011 at a pixel in a region of interest, which may be defined as a hotspot 8003, and actuates a mouse button, the Magic Wand 9011 searches for edges around the pixel 1049 that define a new hotspot 8003 or modify a pre-existing hotspot 8003. The Magic Wand 9011 may use either the rays or flood-fill algorithms 1051, 1053. With the flood-fill algorithm 1053, flooding can be performed based upon either the Red-Green-Blue (RGB) or luminance values of pixels. The flooding may be performed within a tolerance range of RGB color or luminance values around respectively an initial RGB color or luminance value of the pixel. As a result, all pixels surrounding the initial pixel with a RGB or luminance value within the tolerance range will be flooded. The tolerance range can be modified by the user. Flooding may require a line-table hotspot type.

Hotspots 8003 may be added or subtracted from one another to create hotspots 8003 of complex shapes. New hotspots may be created by uniting, or adding, multiple hotspots with one another, subtracting one hotspot 8003 from another, or by intersecting multiple hotspots. The flood-fill algorithm 1053 may also permit incremental hotspot definition by adding newly flooded areas to areas that are already flooded. The flood-fill algorithm 1053 may also permit flooding out areas that are already flooded. The user can interact with the flood-fill algorithm 1053 with a floating menu or the alternate mouse button.

With the rays algorithm 1051, imaginary rays in different directions are launched either outwards from an initial point, or from the hotspot 8003 borders inwards towards the initial point. The rays algorithm 1051 searches along the rays for intersections with the region of interest borders, or edges, that will define the hotspot 8003. The edge points along the rays will be scored using different edge scoring algorithms. Upon completing score, a point on the ray may be selected that either has the best score or is the first point to have score that exceeds a threshold value. When intersection point is found, the intersection point is selected as a new polygon vertex. The number of rays used to define the hotspot 8003 can be modified by the user. The Magic Wand 9011 can also use

edge-enhancing filters 1043 using threshold 1045 and spatial derivative 1047 techniques, prior to using the edge detection algorithms.

Zoom Options

The Workshop window 8001 may permit zooming within a frame to define hotspots 8003 more accurately. This feature may be particularly useful for creating a hotspot 8003 in the shape of a polygon 1041 with, for example, the magic wand 9011 or freehand drawing.

Merging Hotspots

The user can merge multiple hotspots 8003 that may have different shapes into a single hotspot 8003. Hotspots 8003 can be merged over a selected range of one or more frames in a video. Additionally, the user can remove a hotspot 8003 from a group of merged hotspots. The user interface for merging hotspots 8003 can be incorporated into either the Workshop window 8001 or the menu bar. A dialog may be used to enter data pertaining to merged hotspots.

Interpolation

Interpolation 1027 permits a user to quickly create hotspots 8003, for example, in a range of video frames. One embodiment of creating a hotspot 8003 with interpolation 1027 will now be described. First, the user defines the hotspot 8003 in a selected initial, or first, video frame, and presses the Interpolate From button 9002. As a result, the hotspot's color changes. Next, the user defines the hotspot 8003 in a selected last, or second, video frame, after the initial video frame, of the desired range, and presses the Interpolate End button 9004. As a result, the hotspot 8003 is defined by interpolation of size and position in the desired range of video frames. In one embodiment, interpolation 1027 is performed with linear interpolation. The Interpolate From and End button 9002, 9004 may, for example, be part of the Tools window 9001.

The user can also define the hotspot 8003 in one or more selected intermediate frames between the initial and last video frames. As a result, the hotspot is defined by interpolation, such as linear interpolation, between each

successive selected video frame. Also, a hotspot 8003 may be defined by interpolating between two different shapes, for example, in two different frames of a video.

2.142 Hotspot Tracking

Upon defining a hotspot in an initial frame of video, the hotspot can be tracked 1029 through successive frames. Tracking may be performed manually 1035 or automatically 1061. Hotspots can be tracked while the video is playing in reverse.

Manual Tracking

Manual tracking 1035 permits the user to control the placement and size of the hotspot 8003 in each frame of the video. An exemplary method for manual tracking 1035 will now be described. First, the user selects a hotspot 8003 by depressing a (e.g., left) mouse button. Then, while depressing the (e.g., left) mouse button, the user actuates, such as by clicking, the (e.g., right) alternate mouse button. The video will then start playing frame by frame at the specified tracking speed set in the video's Media Properties Sheet. Manual tracking 1035 is performed as long as the user keeps the mouse button depressed. The user toggles 1057 between executing and pausing manual tracking 1035 by respectively depressing and releasing the alternate mouse button. The user can manually move, or slide, 1055 the hotspot 8003 position 1059 about the Workshop window 8001 by moving a mouse, for example, while the video is playing. The user can resize 1059 the hotspot 8003 with the arrow keys while manual tracking 1035 is being performed.

Upon releasing the mouse button, the hotspot 8003 will remain selected. However, the Workshop window mode changes automatically to Pick Hotspot mode 9003. In this mode, the user can resize and move the hotspot 8003. Also, the user can then continue to track the hotspot 8003 either manually 1035 (described above) or automatically 1061.

The user can freely switch between manual and automatic tracking. If the user wants to redefine an existing hotspot, the user can select the Pick Hotspot mode 9003 and again track, manually and/or automatically, the selected hotspot.

Automatic Tracking

Automatic tracking 1061 is activated from the Tools window 9001. A hotspot 8003 in an initial frame must be selected to be tracked. Upon automatic tracking 1061 activation, a new view window is opened and the tracking results are displayed there. Utilizing Win32 multithreading, for example, multiple hotspots 8003 can be simultaneously tracked in separate tracking windows. The hotspot 8003 may then be defined in succeeding frames until the automatic tracking 1061 is halted. When the automatic tracking 1061 is halted, the view window may close and the Workshop window 8001 is updated with the frame at which the automatic tracking 1061 halted.

Automatic tracking 1061 of a hotspot 8003 is accomplished with image-processing algorithms. Both the location, or position, and size of a hotspot 8003 may be automatically tracked 1061 over a range of frames in a video. Location tracking 1073 identifies and follows any changes in the location of the tracked hotspot. Size tracking 1069 identifies and follows any change in the size of the region of interest corresponding to the hotspot 8003. Size tracking 1069 may include a resizing algorithm 1071 to efficiently create resized copies of the frame or the region of interest, which may be defined by a bounding rectangle. Size changes are usually due to a change in perspective like zooming. Size tracking 1069 can be disabled in a Tracking Property Sheet 9101, exemplified in Figure 9B, for example, with a flag 9103.

Automatic tracking 1061 is performed with tracking algorithms 1063, which may include methods of selective enumeration 1065 and scoring 1077. Selective enumeration 1065 may be performed for each frame to determine the new position and size of the hotspot 8003, described above.

Selective enumeration 1065 may be performed for each frame with a steepest, or gradient, descent algorithm 1067 to postulate hotspot 8003 size and position. Selective enumeration 1065 usually avoids enumerating all possible

locations and sizes, thus increasing the speed of the automatic tracking 1065 process. Using the steepest descent algorithm 1067, a path will be found to the hotspot position and size having the best score usually without having to enumerate all possible hotspot 8003 positions and sizes. This technique is efficient for determining hotspot 8003 size and position that greatly differ from the hotspot 8003 size and position in a previous frame.

Automatic tracking 1061 may be stopped and tracking failure may be declared if the selective enumeration 1065 reaches a maximum offset without a minima being found by the gradient algorithm 1067, or if the best score found does not exceed a threshold beyond the average score of all enumerations. To detect when a hotspot 8003 exits a frame, the part of the shape still in the frame may be enumerated with a boundary compensation algorithm 1075.

For each enumerated size and location, scoring 1077 is performed to determine hotspot matching 1079 by correlating the postulated hotspot in the new size and position with the hotspot 8003 in the previous frame, and to recognize tracking failure. Hotspot matching 1079 may measure the correlation, or similarity, of two hotspots in the following way. Shape iteration 1085 may be used to enumerate efficiently the respective pixels in the postulated hotspot and the hotspot in the previous frame, and to calculate the sum of the differences between each set of two corresponding pixels in the postulated hotspot and the hotspot 8003 defined in the previous frame. A smaller sum indicates a high similarity between the postulated hotspot and the hotspot 8003 in the previous frame. The difference can be measured either by calculating the luminance delta 1083 of the corresponding pixels, or by calculating the RGB maximum delta 1081, otherwise known as the maximum norm of the RGB delta values, of the corresponding pixels.

To recognize tracking failure, a simple scoring average may be calculated and compared against a best score. If the best score does not differ dramatically from the average score, the automatic tracking 1061 process will declare a failure. However, normalized scoring using variance and weighted score averaging can also be used. Additionally, the tracking process can be improved by filtering each frame. For example, smoothing or four pixel filters can be used.

Special filters can be assigned to specific tracking algorithms 1063 and scoring 1077 methods.

Scoring efficiency can be enhanced by using the signature 1087 of the tracked hotspot. The hotspot signature 1087 may be a characteristic of the hotspot 8003 in the first frame the hotspot 8003 was defined. For example, the signature 1087 can be the offset between the center of mass 1089 and the geometric center of the hotspot 8003. The RGB or luminance content of each pixel is used to determine the mass of each pixel. To calculate the center of mass 1089, the hotspot's pixels may be enumerated using shape iteration 1085. The benefit of using a signature 1087 is that the score may be calculated by comparing data from the current frame with the signature of any previous frame. Hence, the pixel data of the entire previous frame need not be stored.

Smart scoring can also be used. With smart scoring, the scoring system learns from each tracking step. For example, relying upon temporal coherence 1086 of a moving hotspot, the hotspot motion can be evaluated and new suggested positions and sizes for the hotspot 8003 may be estimated in accordance with the motion trend of the hotspot trends.

Polygons can be automatically tracked 1061 using different techniques, including those described below. First, the bounded rectangle around a polygon can be tracked, and the polygon moved, as is, to the new tracked location. Second, the magic wand 9011 algorithms can be used to adjust the polygon shape after tracking, as described above in the first technique. A third alternative is to use a shape iterator 1085, discussed above, for a polygon, and an inflating or resizing algorithm 1071 to track and directly score the size and position of the polygon.

2.143 Targets

The user can create a hypervideo that is nonlinear 1009 by linking 1010 a hotspot in a source media to one or more targets 7003. A target 7003 may be a playing instance of a media element 1690 including certain parameters exemplified below. Nonlinear authoring 1009 may permit source media and targets to be executed and terminated in a variety of ways, as will subsequently be

described. The hypervideo includes one or more base targets, such as a video target, that are executed when the hypervideo commences. Targets 7003 may, for example, be media elements 1690 or static targets 1022. Media element 1690 targets may include, but are not limited to:

1. **Video 1011** – may be used as hotspot containers from which the user can hyperlink to media. Video targets 1011 (i.e., video media files) are not restricted to any specific format. For example, the hypervideo environment 1000 supports video formats including, but not limited to, Video for Windows (AVI), QuickTime (MOV), motion JPEG and MPEG I and II (MPG) files. Video targets 1011 may, for example, be in any format for which an MCI driver is available.
2. **Sound 1015** – may be used, for example, as a connecting segment between two videos. A sound target 1015 can be a vocal announcement when the user actuates a mouse button when the pointer is over an image, for example, of a person. Supported sound target 1011 (i.e., sound media file) formats include, but are not limited to, WAV, AIF, RMI and MID.
3. **Picture 1013** – may be used as hotspot containers from which the user can hyperlink to media. Supported picture target 1013 (i.e., bitmap media file) formats include, but are not limited to, DIB, BMP, GIF, PCX, TIFF, JPEG and PICT.
4. **Text 1017** – may be used, for example, as a connecting segment between two videos. Text targets 1017 may appear in boxes, such as message boxes. Text targets 1017 can be in, but are not limited to, a TXT or RTF file format, or stored directly in the project file 1670. Text targets 1017 may be displayed with bitmaps also.
5. **Executable 1021** – may be used to launch an executable program (e.g., EXE or BAT file) as a target.

6. **HTML File 1021** – may be opened with a browser (e.g., Internet browser for the World Wide Web) or an OLE control for viewing an HTML file.
7. **URL Addresses 1021** – may be targets created with the authoring tool 1001 and stored as text. Upon activation of this target 7003, a browser, such as an Internet browser, may be launched to access the URL.
8. **QuickTime VR 1021**
9. **Dialogues 1021**

Targets may also include, but are not limited to other applications and processes, communications links to other computers, function menus, high definition television signals, and virtual reality environments.

A static target 1022 is not a media element 1690, but controls a media element 1690 in a hypervideo project. Static targets 1022 include:

1. Back Target – Goes back to the media element 1690 that launched the target 7003.
2. Exit Target – Exits the hypervideo, closes the video window, and returns control to the system or application that started the hypervideo player.
3. Pause Target – Pauses the hypervideo and causes a play button to pop up that enables the user to continue playing the hypervideo by actuating the button.

Other targets 7003, external to the hypervideo authoring software, may also be accessed. External targets may be accessed through a DLL file complying with API requirements to permit an open architecture for the hypervideo environment 1000.

Targets 7003, such as media, can be stored on servers 1655 on local or wide area networks 1650, 1115. Specifically, media, such as video, can be stored on high capacity storage devices such as, but not limited to, digital video, or versatile, disks (DVDs) 1159.

Linking

A user may link 1010 a target 7003 to a hotspot 8003 by the following method:

1. Actuate the mouse button when the pointer is over the Set Target button 9015 in order to enter the Set Target mode. The cursor changes to indicate entrance to the new mode.
2. Select a hotspot 8003 to which the target 7003 will be coupled with the new cursor while not releasing the mouse button. The Set Target movement is a drag-and-drop action.
3. While continuing to hold down the mouse button, drag the new cursor onto the Media Warehouse window and the selected media element 1690 that is to be the target 7003. When the user moves the pointer to a border of the Media Warehouse window, the contents of the Media Warehouse window scroll without releasing the mouse button. The cursor may be changed when a media element 1690 is selected as a target 7003. Depending on the type of target 7003 selected, a Target Properties Sheet 7703, described below, may pop up when the mouse button is released. The user may then enter the properties of the target 7003 into the Target Properties Sheet 7703. Other techniques to link 1010 a hotspot 8003 to a target 7003 may be used.

Target Properties and Multiple Targets

Each hotspot 8003 can have multiple targets 7003. Thus, while playing a hypervideo when the user actuates a mouse button when the pointer is over a hotspot 8003, one or more targets 7003 can be executed or activated. Multiple targets 7003 can be executed sequentially, in parallel or a combination thereof. When the user activates or actuates the hotspot 8003, multiple targets 7003 can be executed in different levels 7601 (e.g., 1, 2, 3, etc.). Levels permit the multiple targets 7003 to be played simultaneously (same level) and sequentially (ascending levels). Each level has a Leader target 7602 which upon ending terminates all other targets 7003 in its level.

Alternatively, the targets 7003 can be positioned and displayed on a time line 7701, exemplified in Figure 7D. As a result, one target 7003 can be initiated during the performance of another target 7003. The time line 7701 can be illustrated on a story board page 7705 of the Target Properties Sheet 7703.

When the user selects a target 7003 for a hotspot 8003, a Target Properties Sheet 7703, which may include a target dialog for the selected media element 1690, may be opened and a new target 7003 is created. The Target Properties Sheet 7703 may include a details page that lets the user enter target parameters. The Details page 7801, exemplified in Figure 7E, may include target 7003 parameters that are unique to the selected media type and permit nonlinear authoring 1009. In the Details page 7801, the user may choose whether the source media, such as video or bitmap, from which the target 7003 was launched, should Pause, Close or Keep Playing 7807 when the target 7003 is executed. The user may also specify how a target 7003 is terminated 7805. Options include:

- **Continue** the hypervideo.
- **Go Back** to the source from which the target originated, skipping subsequent level targets.
- **Loop** – execute the target media in a loop.
- **Freeze** the target's last frame.
- **Exit** the hypervideo.

A second, optional page in the Target Properties Sheet 7703 is a frame window 7901, exemplified in Figure 7F, much similar to the preview page 6001 in the Media Properties Sheet 5003. The Target Properties Sheet 7703 includes, for some of the media types, a Range, or Frame, window that lets the user specify a segment of the target media that will be played. This option may be applicable to video and audio targets 1011, 1015. The frame window 7703, unlike the preview page 6001, lets the user define the IN and OUT frames and times 7903, 7905 for video and audio media files, respectively, to determine starting and stopping frames and times for playing those media files.

The Target Properties Sheet 7703 may also include a Display properties page 7950, for example, for visual media types, including, but not limited to,

video and bitmaps. In the Display properties page 7950, the user may choose how the target 7003 will be projected to the user, as described in the following tables.

Bitmap targets 1013 may have a Place Properties page in their Target Properties Sheet 7703, exemplified in Figure 7G, that permits the user to position a Bitmap over a source element, such as a video.

Upon entering target parameters into the Target Properties Sheet 7703, the user may press one of the following buttons:

1. OK 7811 – Creates a target 7003 for a hotspot 8003 and folds the Target Properties Sheet 7703.
2. Apply 7813 – Creates a new target 7703 for a hotspot 8003. Does not fold the Target Properties Sheet 7703.
3. Cancel 7815 – Closes the last opened Target Properties Sheet 7703.

The following tables describe the different targets 7003 and corresponding parameters. The Sync parameter may not be permitted to be FALSE with any ending parameter setting other than *Continue*.

Video Targets

The following table describes parameters for video targets 1011.

Name	Type	Description
PopWindow	Check Box	Specifies whether the video target 1011 is executed, or played, in the main video window of the hypervideo project or pops up in a new window. TRUE means the target pops up in a new window. Default setting is FALSE.
Move	Check Box	In case a pop-up window is selected, indicates whether the pop-up window should be moved when the user resizes the main video window.
Resize	Check Box	In case a pop-up window is selected, indicates whether the pop-up window should be resized when the user resizes the main video window.

Name	Type	Description
Position	RECT.	When the video target 1011 plays in the pop-up window (PopWindow = TRUE), the user must enter the top left coordinates and the width and height of the new window. The RECT contains the format: left top width height. In case the width and height are absent, the default is the original size of the video.
Sync	Check Box	Specifies whether the hypervideo plays while the video target 1011 plays. This parameter is applicable only if the video target 1011 plays in the pop-up window (PopWindow = TRUE). The default setting is TRUE. When this parameter is FALSE, the Back option for ending the video is not applicable.
In[1]	Button – Marker	This button places a video▾ marker which indicates the first frame to play in a range of frames in a video target 1011.
Out[1]	Button – Marker	This button places a video▴ marker which indicates the last frame to play in a range of frames in a video target 1011.
Ending	Combo Box	Indicates what action to take when the video target 1011 ends playing. The options are: * *Loop* – Loops the video target 1011. * *Back* – Goes back to the source from which the video target 1011 originated. This option is not applicable in case the Sync parameter is set to FALSE. * *Exit* – Exits the hypervideo project player. * *Freeze* – Freezes the hypervideo. * *Continue* – Terminates the video target 1011 and proceeds to the next target. The default option is *Continue*.

[1] The In and Out options are supported by a slider 11,004 that controls the display of the video. The user moves the video to the desired frame and clicks the appropriate button. In case the In marker is placed after the Out marker, the authoring tool 1001 may ask the user whether the markers should be swapped or deleted.

Spatial-Temporal Relative Links

Nonlinear authoring 1009 may also permit targets 7003, such as video targets 1011, to be linked 1010 to temporal 1014, in addition to or in alternative to spatial 1012, coordinates of a hotspot 8003 in a source media. For a temporal relative link 1014, a base frame in a video target 1011, for example, is defined for the first frame in which the hotspot 8003 appears in the source media element. In later frames of the source media element, the hotspot 8003 is linked 1010 to successive frames in the video target 1011 that are temporally related to the base frame. For a spatial relative link 1012, the position, and possibly the size, of the executed, or displayed, video target 1011 is relative to the hotspot's position or to the pointer's position when the hotspot 8003 is activated.

Bitmap Targets

The following table describes some Bitmap target 1013 parameters.

Name	Type	Description
Position	RECT.	The run-time module 1101 needs the top left coordinates and the width and height of the bitmap window. The RECT contains the format: left top width height. In case width and height are absent, the default is the original size of the bitmap.
Duration	Milliseconds	The duration in milliseconds for which to display the Bitmap target 1013.
Sync	Check Box	Specifies whether the hypervideo should stop executing, or playing, until the bitmap target 1013 is no longer displayed. The default setting is TRUE. A FALSE setting does not allow the *Back* option to be set for the Ending parameter.

Name	Type	Description
Ending	Combo Box	Indicates what to do when the bitmap target 1013 is no longer displayed. The options are: *Back* – Goes back to the source from which the video target 1011 originated. This option is not applicable in the case when the Sync parameter is set to FALSE. *Exit* – Exits the hypervideo project player. *Continue* – Terminates the Bitmap target 1013 and proceeds to the next target. This is the only option that lets the user choose a new target after the bitmap is displayed.

When a bitmap target 1013 appears or disappears from the display window, transition effects, including, but not limited to, the following, can be selected. The bitmap may grow from the middle or side of the display window. The bitmap may slide from the side of the display window. The bitmap may evolve from the hotspot. Finally, the bitmap may spirally grow on the display window. These effects can be implemented for videos also.

Audio Targets

Name	Type	Description
Sync	Check Box	Specifies whether the hypervideo plays or pauses while the audio target 1015 plays.
Mix	Check Box	Indicates whether to mix the audio data with other data that is presently playing, or play the audio data "on top" of other data.
Ending	Combo Box	Indicates what to do when the audio target 1015 ends playing. The options are: * *Loop* – Loops the audio target 1015 back and forth. * *Back* – Goes back to the source from which the video target 1011 originated * *Exit* – Exits the hypervideo project player. * *Continue* – Terminates the audio target 1015 and proceeds to the next target. This is the only option that lets the user choose a new target after the audio target 1015 has played. The default is *Continue*.

Audio targets 1015 can include the transition effects of fading in and out.

Mixing of Several Audio Channels

Audio tracks of several different media playing simultaneously may be mixed together.

Text Targets

Name	Type	Description
Position	RECT.	The run-time module 1101 needs the top left coordinates and the width and height of the bitmap window. The RECT contains the format: left top width height. In case width and height are absent, the default is the original size of the bitmap.
Font	Combo Box	The name of the font. Default may be Arial.
Size	Combo Box	The size of the text.
Duration	Milliseconds	The duration in milliseconds that the text target 1017 is displayed.
Infinite	Check Box	Toggle between this parameter and the Duration parameter. When the user specifies the Infinite parameter, the message pops up as a message box and the user may actuate an OK button in order to exit this message box.
Sync	Check Box	Specifies whether the hypervideo should stop and wait for the duration of displaying the text to pass. The default is TRUE. In case the Infinite flag is ON, the Sync flag turns ON too.
Ending	Combo Box	Indicates how to terminate the display of a text target 1017. The options are: *Back* – Goes back to the source from which the video target 1011 originated. *Exit* – exits the hypervideo project player. *Continue* – Terminates the text target and proceeds to the next target. This is the only option that lets the user choose a new target after the text target 1017 has been displayed. The default option is *Continue*.

Text targets 1017 can include the transition effect of evolving letter by letter.

Executable Targets

Name	Type	Description
Command Line	Edit Box	Text describing the command line for the executable target 1021.
Position	RECT.	The run-time module 1101 needs the top left coordinates and the width and height of the executable window. The RECT contains the format : left top width height.
Sync	Check Box	Specifies whether the hypervideo should stop and wait for the duration of opening the executable target 1021 to pass. The default is TRUE.
Ending	Combo Box	Indicates what to do when the executable target 1021 ends execution. The options are: *Back* - Goes back to the source from which the video target 1011 originated. *Exit* - exits the hypervideo project player. *Continue* - Terminates the executable target 1021 and proceeds to the next target. This is the only option that lets the user choose a new target after the executable target 1021 is performed. The default is *Continue*.

Editing Options

The workshop window supports the following editing options:

- Selection – different types are available:

  One hotspot 8003 in one frame;

  Multiple hotspots 8003 in one frame;

  One hotspot 8003 over multiple frames (using the shift key, and the slider or arrow keys); and

  Multiple hotspots 8003 over multiple frames.

- Undo and Redo – for:

  Deleting a hotspot 8003;

  Editing, such as cutting, copying and pasting;

  Defining a hotspot 8003 in more frames;

  Defining a new hotspot 8003; and

  Moving a hotspot 8003.

- Cut, Copy, Paste and Paste New – implemented to work for one or more selected hotspots for one or more frames. A user can cut and copy a selected hotspot and paste it on a different frame. Hotspots 8003 created with the paste command retain their original identity. However, hotspots 8003 created with the Paste New command have new identities.
- Delete and Delete Hotspot – will delete selected or entire instances of a hotspot 8003 in a media element 1690. Deletion can be performed on selected hotspots. When deleting all instances of a hotspot 8003, the hotspot will automatically be deleted throughout the media element 1690. When deleting selected instances of a hotspot 8003 in the authoring tool 1001, the hotspot must remain selected.

Alternate Mouse Button

By actuating the alternate mouse button, different sets of commands can be accessed. The type of command set accessed depends upon whether or not the pointer is over a hotspot 8003. When the pointer is over a hotspot 8003, the alternate mouse button may access the following commands:

- Hotspot Properties
- Go To First Frame
- Cut
- Copy
- Paste
- Delete One
- Delete Hotspot 8003
- Bring To Front (see following Z-order of hotspots section)
- Send To Back (see following Z-order of hotspots section)

When the cursor is not over a hotspot 8003, actuating the alternate mouse button may access the following commands:

- Workshop Properties
- Snap / Stretch
- Different sizes of the Workshop video
- Preview

- Set In Frame
- Set Out Frame
- Go To In Frame
- Go To Out Frame

Z-Order of Hotspots

Hotspots 8003 should maintain a z-order among themselves when one hotspot 8003 overlaps another hotspot 8003. The user may place one hotspot 8003 in back or in front of another hotspot 8003. Selecting a hotspot 8003 will not automatically bring it to the top of the z-order. This option may be supported by one or more of the following portions of the hypervideo environment 1000: the project file 1670, the authoring tool 1001, or the run-time module 1101.

Video Preview

The Workshop window may provide the option to preview a hypervideo being edited in that window.

The Slider

The slider 6007 in the Workshop window permits the user to select and display one frame of a video.

Floating Toolbars

The Workshop Toolbar and Hotspot Toolbar may be floating. The toolbars can hook to the sides of the Workshop window. The orientation of the toolbars may be changed from horizontal to vertical. The toolbars may be displayed in a double vertical format. The interpolation buttons may be placed in the Workshop Toolbar.

Authoring Tool GUI

Figure 10 illustrates an exemplary graphical user interface (GUI) 10,001 of the authoring tool 1001. The authoring tool's GUI 10,001 includes the menu bar 10,005, the main tool bar 10,009, the Media Warehouse window 10,003, the

Workshop window 8001, and the Tools window 9001. A hypervideo 10,007 is shown in the Workshop window 8001.

2.15 Project View

The project view 1007 illustrates the hypervideo story board 1116, shown in Figure 11, which illustrates the base target 11,001 and targets 7003 linked 1010 to hotspots 8003. Icons 11,003 illustrating a thumbnail illustration 4007 of hotspots 8003 in the targets 7003, 11,001 are appended to the corresponding base target 11,001 and other targets 7003. Targets 7003 corresponding to the hotspots 8003 illustrated by the icons 11,003 may then be appended to the icons 11,003. The branch 11,005 of an icon 11,003 that has already been displayed in the story board 1116 may terminate. Branches 11,005 to targets 7003 of newly displayed icons are shown. In this way, the user can view the hotspots 8003 and targets 7003 of all media elements 1690 in the hypervideo project.

2.2 Run-Time Module

The run-time module 1101, illustrated in Figure 1B, is used to play one or more hypervideos simultaneously. The run-time module 1101 utilizes the data in the project file 1670. The run-time module 1101 can be used to play hypervideos 10,007 with different applications, including, but not limited to, Director5 by Macromedia Incorporated (San Francisco California), Shockwave by Macromedia Incorporated, VDOLive by VDOnet Corporation (Palo Alto, California), Netscape Navigator by Netscape Communications Corporation (Mountain View, California), and Internet Explorer by Microsoft. By communicating with the run-time library 1113 through player interfaces 1102, the different applications use the run-time module 1101 to play hypervideos 10,007.

The player interfaces 1102 include, for example:

(1) plug-ins 1103 for applications such as those described above (e.g., Director5 and Netscape);

(2) OLE control, such as ActiveX (OCX), 1105, for example, to integrate with Visual Basic environments by Microsoft Corporation;

(3) stand-alone modules 1107 such as a stand-alone player incorporating the player interface 1102;

(4) an MCI driver 1109 accessible by MCI; and

(5) a software development kit (SDK) 1111.

The run-time library 1113 permits the hypervideos 10,007 to be played. The run-time library 1113 may include four elements, exemplified below:

(1) Execution of a hypervideo project by a project interpreter 1117. The project interpreter 1117 manipulates an object database 1127. The object database 1127 may include an object tree which will be subsequently described. Logging information and preparing statistics 1147 pertaining to the hypervideo project execution can be recorded.

(2) Administration including manipulation, for example, of media necessary for the execution of a hypervideo project is provided by a media manager 1125, including an active media stack 1141, a cache 1143 of media that have been previously activated, and a look ahead system 1145 containing media that are expected to be activated. The media manager 1125 is used to manipulate media files, e.g., opening, playing and closing media files 1680. The media manager 1125 reduces the time to open a file, by keeping track of the opened files and using the look-ahead system 1145 to load new media files 1680 that may potentially be requested by the user. The newly loaded files are added to the object tree, subsequently described.

(3) Manipulation of the hypervideo project, for example, including the object database 1127, can be achieved through an interface 1119 by activation feedback 1129, and display and sound mechanisms 1131 Activation feedback 1129 permits control of the hypervideo project by user or programmatic actions 1149, 1151. Programmatic actions 1151 may be implemented in software, such as by the use of scripting languages 1004. A programmatic action 1151 may include the use of notification frames 1157, subsequently described. A user action 1149 may include actuating a hotspot 8003, while a hypervideo project is being displayed, by, for example, actuating a mouse button, when the pointer is over the hotspot. As a result, a target 7003, such as a media element 1690, may be executed.

(4) Network communications capability 1115, for example on the Internet or an Intranet, that permits implementing the run-time module 1101 in a client-server architecture 1121. The client-server architecture 1121 may permit the run-time module 1101 to support multiple users 1133. The client-server architecture 1121 may also permit the run-time module 1101 to support streaming of video and other meta-information. A queue manager 1137 manages a stream of information, including hypervideo information, being transmitted between the client and the server 1121.

2.21 Run-Time Module Commands

The run-time module 1101 may support the following commands:

- Open project – Loads a specified hypervideo project (.OBV) file 1670. This operation may also modify the display window of the hypervideo project according to submitted window parameters. The window parameters include window size, position, z-order and hierarchical status.
- Play project – Activates a previously loaded hypervideo project. A call-back object 12,001 may be supplied when using this command.
- Stop project – Stops a playing hypervideo project.
- Pause project – Pauses a playing hypervideo project.
- Set Project Window Properties – Window properties of an open project may be altered. These properties include window size, position, z-order and hierarchical status.
- Add \ Remove A Call-Back Object – Call-back objects may be added or removed.

2.22 Run-Time Module Design

An exemplary design of the run-time module 1101 will now be described. This description will include an illustrative discussion of call-back objects, a player interface 1102, and the run-time library 1113.

Call-back Objects

An application may include a call-back object 12,001, exemplified in Figure 12. Call-back objects are otherwise known as call-back windows or call-back functions. The call-back objects 12,001 couple the run-time library 1113 to an application 12,003, which may include the stand-alone module 1107 or other applications described above. Specifically, the call-back objects 12,001 facilitate communications between the run-time library 1113, and the application 12,003. A call-back object 12,001 receives information, in the form of messages or function calls, upon the occurrence of certain events when a hypervideo 10,007 is performed. The call-back object 12,001 communicates the information to the application 12,003. The events are listed in the following section describing Event Notification 1114. Furthermore, the application 12,003 can initiate actions in a hypervideo 10,007 through a call-back object 12,001. The use of the term object in this paragraph and subsequently in section 2.2, refers, not to hotspots 8003, but to objects in object oriented programs.

Event Notification

The run-time library 1113 supports notification of call-back objects 12,001 upon the occurrence of certain events (1114). Upon such event notification 1114, call-back objects may present certain return values to the run-time library 1113, or initiating actions of the type listed below. Events include:

- Hotspot related events – these events include:
  - * Entrance of a hotspot 8003 into a frame that is the first frame in which the hotspot 8003 is defined.
  - * Exit of a hotspot 8003 from a frame that is the first frame in which the hotspot is not defined.
  - * Activation of a hotspot 8003, when a user activates a mouse button when the pointer is over the hotspot 8003 in a certain frame.
  - * Passing over a hotspot 8003, when the pointer is moved over a hotspot 8003. Hotspots may respond to this notification, for example, by specifying a cursor to be displayed.

Call-back objects 12,001 may respond to any one of these hotspot-related events.

- Media Related Events – These events include:
  - * Start or end of a media, when a media is activated or deactivated. The start and end of the media may not be the first and last frames of the media, but rather the first and last frames of the media that are displayed.
  - * Reaching certain marked frames or other previously marked points. Certain frames of a media may be marked during hypervideo project authoring, or during run-time, as notification frames 1157. When notification frames 1157 are executed or displayed, the run-time library 1113 notifies the call-back objects 12,001.
- Target Related Events – Specific targets may be marked for notification. Before activating, or executing, such targets 7003, call-back objects 12,001 are notified. This notification allows the hypervideo 10,007 to abort the target 7003 execution or commence different related actions.

Initiated Actions

When a hypervideo 10,007 is playing, applications 12,003 which reference the hypervideo project may initiate two kinds of actions:

1. Actions that have a direct and apparent influence on the playing hypervideo 10,007. These actions include:
   - Target 7003 Activation – whereby an application 12,003 can force the playing hypervideo 10,007 to launch, or execute, a certain target by creating a new target 7003 or by activating, or actuating, a hotspot 8003 linked 1010 to a previously defined target 7003.
   - Project Stop – whereby an application 12,003 can force a playing hypervideo 10,007 to stop.
2. Actions directed at the project file 1670 that have an indirect influence on a playing hypervideo 10,007. These actions include:
   - Changing cursors – an application 12,003 may change the cursors associated with a particular hotspot. These cursors include

cursors that are used when a pointer is passed over the hotspot 8003 or when a mouse button is actuated when the pointer is over the hotspot 8003.

- Enabling or disabling hotspots 8003 – an application 12,003 can disable or enable a hotspot 8003. A hotspot 8003 can be fully or partially disabled. With full disabling, a defined hotspot 8003 is completely ignored. With partial disabling, specific hotspot 8003 functions are disabled, such as predefined targets 7003, notification, and hotspot visualization 1155, for example.
- Replacing the base targets 11,001 – whereby an application 12,003 can redefine the hypervideo base targets 11,001 which are played when the hypervideo 10,007 begins. This function may be invalid once hypervideo 10,007 has commenced playing.
- Marking notification frames 1157 – whereby at run time, the application 12,003 marks frames for which notification is requested.
- Querying the media and media position – whereby an application 12,003 can retrieve at any given time the currently playing media(s) and its/their current position(s) (e.g., frame or time).

MCI Driver

The run-time module 1101 may include an MCI Driver 1109 as one of the player interfaces 1102 to permit the playing of hypervideos 10,007. The MCI driver 1109 may allow Window applications to play hypervideos 10,007, using the Multimedia Control Interface (MCI) by Microsoft. An application 12,003, requesting to play a hypervideo 10,007, may send the appropriate MCI commands. The MCI may translate these commands into messages and send them to the MCI driver 1109 corresponding to the file type that is being acted upon. The corresponding MCI driver 1109 may recognize these messages and direct the run-time library 1113 appropriately. The MCI driver 1109 may also process user input, for example, from a mouse or a keyboard.

In order to respond correctly to both sources of input, the MCI driver 1109 includes the following two components:

(1) A driver procedure which responds to relevant messages sent by the system; and

(2) A window procedure attached to a window. This procedure is designed to trap relevant mouse messages, and to serve as a call-back object 12,001 for notifications sent by the playing media elements 1690.

In order to accomplish their tasks, the components use a set of hypervideo objects created and used by the run-time library 1113. Any time a target 7003 has to be executed as a result of activation feedback 1129, an object is constructed. These objects include information about a corresponding target 7003. The objects that are playing or that were playing and may be played in the future, are kept in an object tree. In this object tree, objects that were activated from other objects are depicted as their descendants. Figure 13 illustrates exemplary objects 13,002 in an exemplary object tree 13,004. The objects 13,002 and object tree 13,004 may also be used with player interfaces 1102, including the MCI driver 1109, described above. The MCI driver and window procedures will now be described in more detail to exemplify the use of objects 13,002 and the object tree 13,004.

MCI Driver Procedure

The MCI driver 1109 may be designed to respond to the messages sent to it by the system. These messages may include general driver messages, and MCI-specific messages. The MCI messages are interpretations of the messages sent by an application. The messages handled by the driver include: MCI_OPEN, MCI_PLAY, MCI_STOP, MCI_PAUSE, MCI_RESUME, MCI_WINDOW, MCI_CLOSE, MCI_PUT, and MCI_UPDATE. Custom messages may be developed to enable seeking specific clips in given frames or targets 7003. An exemplary description of some of the messages follows:

MCI_OPEN (relevant parameters: Open-Flags, Open-Parms):

This message initiates the hypervideo 10,007 according to the file name found in the Open-Parms. When processing this message, the base target 11,001 to be played becomes the root object 13,006 of the object tree 13,004. According to the Open-Flags and Open-Parms, a window is created. The window's procedure is set to be the driver's WndProc.

MCI_PLAY:

This message starts playing the hypervideo 10,007. The message can be received, for example, upon one of the following situations arising: (1) The project is stopped – In this case the object tree 13,004 is initiated from the root object 13,006. The base target 11,001, corresponding to the root target 13,006 is thus played. (2) The project is paused - In this case all the objects 13,002, or corresponding targets 7003, which have the status flag Playing (e.g., were previously playing, see MCI_PAUSE) are played.

MCI_STOP:

The hypervideo 10,007 and all of its playing targets 7003 are stopped, and the object tree 13,004 is collapsed.

MCI_PAUSE:

The hypervideo 10,007 and all of its playing targets are paused. Pausing does not affect the status flag of the objects 13,002. Thus, when resuming the playing of the hypervideo 10,007, the run-time module 1101 knows which objects 13,002 to resume playing. There is no effect on the object tree 13,004.

MCI_RESUME:

This message is effective only if the hypervideo 10,007 is paused. In this case all the objects 13,002 of the hypervideo 10,007 marked Playing are played.

MCI_WINDOW (relevant parameters: window):

This message replaces the window relating to the 'full-screen' of the MCI driver 1109. It can be sent only when the MCI driver 1109 is not playing. The window is attached to the root object 13,006, and is subclassed to WndProc.

MCI_CLOSE:

This message releases all memory, and terminates the MCI driver 1109.

Window Procedure

Each driver session is associated with a window. The association is made by means of the MCI_OPEN and MCI_WINDOW commands. The association must be done before playing starts. The driver uses the window procedure to obtain mouse and MCI events. In case the driver is assigned a new window using the MCI_WINDOW command, it will subclass its window procedure. Any other child windows which are created during the playing of the hypervideo 10,007 are also subclassed to this same procedure, and the interaction with them is done through it.

The main messages handled by the window procedure are:

WM_LBUTTONDOWN (relevant parameters: window, mouse position):

This message is used to identify user action of actuating a mouse button. From the window handle, the object 13,002 can be retrieved via the management manager 1125. After identifying the object 13,002 and the pointer coordinates, the object database 1127 is queried for a target 7003, or object 13,002, to be executed. According to the data received from the object database 1127, the current object 13,002 may be stopped.

WM_SETCURSOR (relevant parameters: window):

This message is used to set the cursor according to the pointer coordinates, whether the pointer is over a hotspot 8003, whether a mouse button has been actuated, and depending upon the media element 1690

being executed. From the window handle, the object 13,002 can be retrieved via the management system.

MM_MCINOTIFY (relevant parameters: device ID):

This message is used to signal that an object 13,002 has terminated. From the device ID, the object 13,002 is retrieved via the media management system 1125. Notification of medias that terminated unsuccessfully may be ignored at this stage. The object 13,002 that ended is updated to show that the corresponding media file 1680 is no longer playing. For compound objects, this may not necessarily mean that the object 13,002 has terminated. If the object 13,002 is terminated, then a check is performed on the object's termination flag, or ending parameter, previously described. The following actions are taken according to the following flag settings:

if EXIT stop all playing target(s) 7003.

if LOOP play target(s) 7003 again.

if CONTINUE stop target(s) 7003 and take it/them off the active media stack 1141. Get the next object 13,002 and play it.

if BACK stop target(s) 7003 and take it/them off the active media stack 1141. Continue the parent object or target.

if FREEZE pause target 7003.

MM_MCISIGNAL (relevant parameters: device ID, video position):

This message is used to track a playing object 13,002. Again, the object 13,002 may be retrieved from the device ID. Tracking a playing object 13,002 permits the following:

1. Since the image in a video, for example, may change, the cursor may have to be altered even though the mouse has not moved. For that reason, it may not be sufficient to wait for the WM_SETCURSOR message.
2. Identifying notification frames 1157. For each frame, the object database 1127 is queried to determine if the frame is a notification

frame 1157. If the frame is a notification frame 1157, the corresponding target(s) are executed.

3. Logging, for example in a database, the entrance, or appearance, and exit, or disappearance, of hotspots 8003.

An object 13,002 always keeps its position, for example frame number, of its last signal message. When such a signal message is received, the object 13,002 is queried by itself for changes that occurred between the last kept and current position.

WM_TIMER (relevant parameters: device)

This message is used to signal that a timed object has terminated. The action is similar to the notify message.

Objects and the Object Tree

The object 13,002 includes a description of a corresponding target 7003. Additionally, the object also may include status information, such as playing status, the MCI device ID and relevant window.

The objects 13,002, or corresponding targets 7003, that are being executed, or that were executed and still may at anytime in the future be executed may be kept in an object tree 13,004 The objects 13,002 of the object tree 13,004 may be of two general types: visible 13,008 or invisible 13,010. The visible objects 13,008, unlike invisible objects 13,010, can serve as parents of other objects 13,002. Invisible objects 13,010 are always leaves.

An object 13,002 may be added to the object tree 13,004 once the object 13,002 starts playing. The object 13,002 is removed from the object tree 13,004 once the object 13,002 is no longer scheduled to be played. An object 13,002 may be added to the object tree 13,004 in the following situations:

1. When starting to play the hypervideo 10,007, a root object 13,006, representing the base target(s) 11,001, is created and is placed at the top of the object tree 13,004.

2. As a result of actuating a switch, or button, on a mouse or reaching a notification frame 1157, an object 13,002 is spawned from the object

13,002 that contains, for example, the hotspot 8003 over which the pointer was placed when the switch was actuated, or clicked.

3. If an object 13,002 ends or is closed, and its termination flag is set to continue, the closed object 13,012 is destroyed and a new object 13,014 is placed as a son of the closed object's parent object.

An object 13,002 may be removed from the object tree 13,004 in the following situations:

a. When the hypervideo 10,007 reaches its end or the close command is received, the object tree 13,004 is terminated.

b. If an object 13,002 terminates and the termination flag is set to continue (used to activate new object 13,014) or to back (used for asynchronous objects to continue playing their parent object).

Objects 13,002 removed from object tree 13,004 may be marked in one of two ways: (1) stop all the playing descendant object(s) or (2) do not stop all playing descendant object(s). If an object 13,002 of the first type (1) is removed from the object tree 13,004, then descendent object(s) that branch from the removed object are also removed from the object tree 13,004. If the object 13,002 of the second type (2), is removed from the object tree 13,004, then branch(es) of descendent object(s) of the removed object are appended to the parent object of the removed object.

Media Manager

The media manager 1125 includes a list of media file 1680 names. The media manager 1125 also includes an MCI device ID for each media file 1680 in the list. The media manager 1125 further includes a list of objects 13,002 that use those device IDs. As a result, the objects 13,002 can interact with the MCI.

The media manager 1125 supports the following commands:

Open (media file, object reference, window) – this command opens a media file 1680 in a specified window and returns the MCI device ID of the opened media file 1680. If a copy of the specified media file 1680 has previously been opened and is available, the same media file 1680 copy and MCI device ID will be used.

Stop (object reference) – stops the playing of the media file 1680 relating to the object reference.

Close (object reference) – removes the object.

Database Resolution

Resolution of a hypervideo project may be 1000 by 1000. To optimize the speed of database calculations, the resolution may be 1024 by 1024. Also, the user can change the resolution of the hypervideo project coordinates.

Pointer Location

The Microsoft Foundation Class (MFC) CRgn objects may be used to determine whether the pointer is positioned over a polygon-shaped hotspot.

Overlaying Displayed Media Files

Two or more media files 1680 may be displayed overlapping each other. A z-order relationship is maintained between the media files 1680 so that they are displayed over one another in a consistent fashion. Special overlay techniques, such as key color, blue screen and alpha channel may be used to achieve sprite animation and non-rectangular overlaps of windows.

Hotspot Visualization

A marker 1155 can be attached to a hotspot 8003 (i.e., the hotspot 8003 can be marked). The marker 1155 can serve two purposes: First, the marker 1155 signals the user that a hotspot 8003 exists. Second, the marker 1155 can signify that the user has already actuated the hotspot 8003.

To facilitate these purposes, the marker 1155 can be attached to the hotspot 8003 in one of the following ways:

- Statically by showing the marker continuously. The marker 1155 can be turned on or off by an application 12,003, such as a monitoring application; or

- Dynamically when, for example, the hotspot 8003 is actuated by an application 12,003, such as a monitoring application, or when a pointer is over the hotspot 8003.

The marker 1155 may be implemented using one of the following techniques:

(a) A text tag attached to the hotspot 8003;

(b) A bitmap tag attached to the hotspot 8003;

(c) Overlapping the hotspot 8003 with a surface, such as a translucent surface to highlight the hotspot 8003;

(d) With sprite animation; and

(e) With tool tips. When a mouse moves over a hotspot 8003, a tool tip may pop up to specify, for example, the hotspot's name.

Run-time Statistics

Statistics 1147 are gathered for hotspots and media files 1680. Statistics 1147, for example, include the number of times a hotspot 8003 has been activated or the number of times a particular media file 1680 was played. The statistics 1147 can be accessed used by applications 12,003, such as monitoring applications, to initiate conditioned actions. The statistics mechanism can be replaced, at least partially, by a counting mechanism in the application 12,003 that is triggered by event notification 1114. Additionally, statistics 1147 can be displayed.

Project Status

The current status of a project, including currently playing media, for example, defined by the object tree 13,004 and those waiting in the active media stack 1141, can be saved by the run-time library 1113. The saved status can later be recalled to continue playing a project from a specific point.

2.3 Streaming Hypervideo

Hypervideo data can be streamed, as described above, for example across a network 1650 from a server to a client, as described above. Because the hypervideo data is streamed, instantaneous network bandwidth consumption is diminished. Also, latency, associated with hypervideo performance, may be reduced. Techniques for streaming hypervideo data will now be further described.

Media Objects

In one embodiment, media objects may be hotspots 8003 in visual media, such as a videos and bitmaps, that may or may not be associated with interactive features, such as links to targets 7002. In another embodiment, media objects, however, may be associated with other hypervideo features such as targets 7003 (or links), cursors (e.g. click and over cursors), and markers 1155.

Media objects may be referenced in an object oriented manner, in addition to a traditional image oriented fashion, such as by reference to a frame or time base. Because media objects may be referenced in an object oriented manner, media objects may be referenced in a relatively natural manner. Not only is interactivity permitted with media objects, but so also is storage and retrieval of media objects (or object based indexing of media), and media object statistics.

Streaming of Media Objects

In one embodiment, media object data may be included in streaming multimedia. In another embodiment, media object data may be included in the hypervideo data stream 1660.

In a further embodiment, the multimedia stream may include a video and audio stream. In yet another embodiment, the video stream may include a variety of visual information including an image, a slide show, and/or an animation.

The hypervideo data stream 1660 may permit a user to interact with a video stream in a manner similar to that described above. In one embodiment, the

hypervideo data stream 1660 may include information, such as geometry and shape, about a media object in a frame of a video.

Hypervideo data may be extracted from the hypervideo data stream 1660, and may be provided to a hypervideo application, with an application program interface (API), as will be further described below. Streaming hypervideo may be displayed and accessed by a user by modifying the run-time module 1001 to include a hypervideo data stream 1660 renderer. The hypervideo data stream 1660 renderer may include an API that provides access to media object information in the hypervideo data stream 1660, and may permit complex interaction with media objects.

To diminish latency and instantaneous network bandwidth consumption, the hypervideo data may be streamed with the streaming multimedia. The hypervideo data may, however, be downloaded prior to the streaming of the multimedia. However, this downloading approach may undesirably increase latency and instantaneous network bandwidth consumption which may scale, for example linearly, with the length of the streaming media.

Streaming hypervideo data may also be desirable to accommodate the dynamic nature of a video. Video may be dynamic in nature. For example, the geometry of the media object may change from frame to frame, for example, due to variations in perspective. If, however, the geometry of the media object in the hypervideo data stream 1660 is fixed over a range of frames, the media object geometry may not be accurately defined in the range of frames. As a result, user interactivity with the hypervideo 10,007 may be undesirably diminished. Therefore, in order to enhance user interactivity, media object data in the hypervideo stream may be dynamically provided, for example for each frame in a corresponding video. One technique for streaming hypervideo data for each frame of a corresponding video will now be described.

Streaming Hypervideo Data For Each Video Frame

The following technique may implement streaming hypervideo by providing hypervideo data for each frame of a corresponding video. This technique has the advantage that the hypervideo data stream may be randomly

accessed at any time in the stream. Further, this technique may used with no, or a relatively small, hypervideo data stream header so that less memory is required of a recipient client to store such a header.

Further, this technique may permit manipulation of hypervideo data streams 1660, and exportation of hypervideo data into a hypervideo data stream 1660 of a streaming multimedia file. In one embodiment, the streaming multimedia file may be an audio video interlace (AVI) file. A hypervideo data stream 1660 may be stored in a private track of an AVI file. This embodiment may be implemented with computer software, such as file Hvstrvw.cpp, illustrated below. The operation of the computer software will now be described.

A class ChvstreamView may include a method OnStreamAddstream. In one embodiment, when invoked from an object instantiated from class ChvstreamView, method OnStreamAddstream may export hypervideo data from a project file 1670 through an authoring tool 1001 to a private track of a designated AVI file. This exportation process may be repeated for each AVI file in a hypervideo 10,007.

The method OnStreamAddstream will now be described. A designated AVI file may be opened for reading and writing. Then, the streaming video track in the AVI file may be accessed. Information from the streaming video track may be read. This information may be used, in part, to create the hypervideo data stream 1660 stored in the private track of the AVI file. In one embodiment, the information from the streaming video track may be used to create the header 1402 and body 1404, illustrated in Figure 14, of the hypervideo data stream 1660. The body 1404 may be adjacent, or otherwise coupled, to the header 1402.

The body 1404 of the hypervideo data stream, which may be stored in the private track, may be created in the following manner. The hypervideo data (e.g. hotspot data) for each frame in the AVI file may be extracted and stored in the private track. For example, every hotspot 8003 associated with, or in, each frame of the AVI file may be identified, and corresponding hotspot 8003 data may be extracted. The hotspot 8003 data corresponding to each frame may be stored, in the body 1404 of the private track, on a frame by frame basis. The hotspot data

may be stored in an order corresponding with the order of the video frames. Then, the AVI file may be closed. All AVI files in the hypervideo 10,007 may be so modified. In one embodiment, this and other techniques for creating files with streaming hypervideo data may be implemented with modified authoring tools, further described below.

Because hypervideo data is provided for each frame of a corresponding video, this technique may consume a relatively large amount of network capacity. Thus, this technique may be relatively costly. Therefore, it may be preferable to use a more efficient technique for streaming hypervideo data.

Hypervideo Data Stream with a Header and a Body

Network channel capacity may be reduced by placing static hypervideo data in the header 1402 of the hypervideo data stream 1606. For example, static hotspot data may be placed in the header 1402 because the static hotspot data does not vary throughout the hypervideo 10,007. Because it is placed in the header 1402, static hypervideo data may be transmitted once, unlike in the technique described above, reducing transmission bandwidth consumption.

Static hotspot data may include data about the click and over cursors, a marker 1155 and target(s) 7003 defined for a hotspot 8003 because such data may remain constant throughout the life-time of the hotspot 8003. Specifying such attributes more than once for each hotspot 8003 (*e.g.* for each frame in which the hotspot 8003 occurs) may be undesirably redundant as described above.

This streaming technique, like the previously described technique, facilitates the exportation of hypervideo data into a hypervideo data stream 1660 of a streaming multimedia file. In one embodiment, the streaming multimedia file is an audio video interlace (AVI) file. The hypervideo data stream 1660 may be stored in a private track of the AVI file. In one embodiment, this embodiment may be implemented with computer software, e.g. file ExportAvi.cpp illustrated

below. The operation of the computer software will now be described.

In one embodiment, file ExportAvi.cpp includes the following arguments:

a_sProjectFile which is the name of the hypervideo project file 1670 which includes media file(s);

a_sMediaName which is the name of the AVI file that includes the audio and video streams to be incorporated into a new AVI file; and

a_sTargetAVI which is the name of the new AVI file in which the streaming hypervideo data 1660 will be written on a private track.

The computer software operates in the following manner. A project file 1670 may be opened. In one embodiment, the data in the project file 1670 may be encrypted.

Hypervideo data may be extracted from the project file 1670 identified by the a_sProjectFile argument. In one embodiment, the extracted hypervideo data may be stored in an EProject object. An EProject object may be an object oriented representation of the database in which hypervideo data is stored, for example in the authoring tool 1001. The EProject object is further described below.

The hypervideo data may be extracted from the project file 1670 in the following manner. The amount, or length, of data in the project file 1670 may be determined. A buffer, having such length, may be allocated. Encrypted data is read from the project file 1670, and may be stored in the buffer. The encrypted data stored in the buffer may be decrypted. A memory file may be created, and attached to the buffer. Memory files and buffers are well known to persons skilled in the art. An archive may be attached to the memory file. The data in the buffer may be transferred to the EProject object. The archive and project file 1670 may be closed.

Assuming that the extraction process is successfully completed, hypervideo data associated with the selected AVI file, in the hypervideo and identified by the a_sMediaName argument, may be transferred to a new, corresponding AVI file, identified by the a_sTargetAVI argument. The hypervideo data may be written in the private track of the new AVI file. The

hypervideo data may be written in the private track of the new AVI file. The hypervideo data transfer may be performed in the following manner.

Streamed hypervideo data may be extracted from the EMedia object that corresponds to the selected AVI file. An EMedia object may represent, in an object oriented manner, the AVI file and its associated hypervideo data. The EMedia object corresponding to the selected AVI file may be selected in the following manner. EMedia objects may be iteratively analyzed to uncover the EMedia object that is associated with the selected AVI file.

Upon ascertaining the EMedia object corresponding to the selected AVI file, the new AVI file may be prepared. If the new AVI file does not exist, then the new AVI file may be created, including by copying the streaming audio and video data from, the selected AVI file to the new AVI file. If the new AVI file already exists, but does not include the same video stream as the selected AVI file, then the selected AVI file's audio and video streams may be copied to the new AVI file.

The file ExportAvi.cpp includes method AddObjectInformation which creates a hypervideo data stream 1660 in the private track of the new AVI file. This function includes the arguments from the EProject object (corresponding to the selected project file 1670) and the EMedia object (corresponding to the uncovered EMedia object, and including hypervideo data and pointer to the selected AVI file), and the name of the new AVI file. The method may operate in the following manner.

Video For Windows (TM) software library may be initialized. A pointer, in the uncovered EMedia object, to an EClip object may be acquired. The class EClip, derived from the class EMedia, may include parameters for storing corresponding hypervideo data. In one embodiment, the classes EClipObject and EClipFrame, derived from class EClip, may include parameters for respectively storing static hotspot data (e.g. target, cursor, marking, and frame range data) and dynamic hotspot data (e.g. geometry and shape data).

In one embodiment, the static hotspot data may be acquired in the following manner. The number of frames in the video stream of the selected AVI file may be determined. Static hotspot data may be extracted for each hotspot by

indexing the EClipObject objects associated with the EClip object corresponding to the selected AVI file. The extracted static hotspot data for each EClipObject object may include data about a click cursor (1 byte), an over cursor (1 byte), a Target URL (1+MAX_URL byte), and a hotspot object frame range (2 DWORD = 2 * 4 bytes). In other embodiments, the static and dynamic hotspot data may include parameter data different than described in this section. The static hotspot data may be stored in the allocated memory to form the header 1402 of the hypervideo data stream 1660 in the private track of the AVI file. In one embodiment, each set of static hotspot data in the header 1402 may correspond to a unique instance of a hotspot object. Other data obtained from the video stream of the selected AVI file may also be used to form the header 1402.

In another embodiment, the dynamic hotspot data (e.g. hotspot geometry, shape) may be similarly acquired by indexing the EClipFrame objects in the EClip object corresponding to the AVI file. The dynamic hotspot data may be reorganized by frame, rather than by hotspot, and copied to the allocated memory to form the body 1404 of the hypervideo stream data in the private track of the AVI file. In one embodiment, dynamic hotspot data may be provided for each video frame. The dynamic hotspot data may be stored in the allocated memory to form the body 1404 of the hypervideo data stream in the private track of the AVI file.

If a frame of video is identified as key frame in the video stream of the selected AVI file, the corresponding hotspot data associated with the key frame is identified as key hotspot data in the hypervideo data stream 1660. Upon creating the hypervideo data stream 1660 in the private track of the AVI file, the AVI file is closed.

Hypervideo Data Stream Having Dynamic Hotspot Information Provided for Multiple Video Frames

A further technique for diminishing network capacity consumption will now be described. This technique permits the hypervideo data rate to be reduced below the corresponding video frame rate. Unlike the previously described techniques, which required one set of streamed hypervideo data to be uniquely

associated for each video frame, this technique permits one set of streamed hypervideo data to be associated with one or more video frames.

In one embodiment, when the set of streamed hypervideo data is received by the client in a client-server network, the geometry data, in the streamed hypervideo data, may be associated with each frame of the one or more video frames. In another embodiment, the geometry data associated with each frame of a set of more than one video frames may be determined by interpolating the geometry data of two consecutive sets of streamed hypervideo data sent respectively before and after the corresponding set of more than one video frames.

Hypervideo data may be exported to a private (proprietary) stream in media file formats, including an ASF version 1 file, using hypervideo data buffers. Hypervideo data buffer streaming will be first discussed. Then, a technique for implementing hypervideo data in streamed hypervideo data buffers will be exemplified for a Netshow (TM) client-server system.

Hypervideo Data Buffer Format

In one embodiment, hypervideo data buffers may contain dynamic media object data about hotspots, including data about hotspot geometry, target(s) 7003, and cursors 7003. Hypervideo data buffers may also include media object data for notification frames 1157, or hotframes, including target 7003 data.

A hypervideo data buffer may include data about media objects associated with one or more video frames. When exporting a hypervideo 10,007 data to an ASF version 1 file, hypervideo data buffers may be created at user defined rates. The maximum hypervideo data buffer creation rate may be limited to a corresponding video's frame rate so that the maximum number of hypervideo data buffers equals the number of frames in the video.

In one embodiment, hypervideo data buffers may be implemented with computer software, e.g. file NSCommands.cpp, illustrated below. File NSCommands.cpp may facilitate the creation of, and the extraction of hypervideo data from, hypervideo data buffers. Object oriented programming classes defined and implemented in file NSCommands.cpp may be

used in an authoring tool 1001 to create hypervideo data buffers. These and other classes may be defined in file NSCommands.cpp, for use in the run-time module 1001, to extract hypervideo data from hypervideo data buffers. File NSCommands.cpp includes the following three classes:

ENSBuffer: Objects instantiated from this class read data from and write data to hypervideo data buffers. For example, this class's method 'Int' writes an integer to or reads an integer from a hypervideo data buffer depending upon the setting of 'storing' flag (e.g. if a hypervideo data buffer's 'storing' flag equals true the integer is written in the hypervideo data buffer when this method is performed).

ENSFrameInfo: Objects instantiated from this class handle the creation of a hypervideo data buffer.

ENSFrameInfoManager: Objects instantiated from this class manage the process of hypervideo data buffer creation. Upon exporting a hypervideo 10,007 to a streaming media file, an object instantiated from the ENSFrameInfoManager class is called repeatedly according to the hypervideo data buffer creation rate every time a hypervideo data buffer is created. Each time an object instantiated from the ENSFrameInfoManager class is called, it receives an ENSFrameInfo object and an ENSBuffer object, and serializes the ENSFrameInfo object to the ENSBuffer object.

ENSFrameInfo Class

Objects instantiated from the ENSFrameInfo class may include information about media object geometry and target(s) 7003 for a frame in the video. Objects instantiated from the ENSFrameInfo class may include an array of ENSHotspot objects, which includes location and target data for each hotspot in a frame, and an ENSEvent object which includes target data for the frame if the frame is a notification frame 1157. Both ENSHotspot and ENSEvent classes inherit properties from a ENSFrameLinkData class which also includes parameters that describe target(s) 7003. A set of these parameters may be used to store Boolean variables for selecting target type. Hotspot 8003 and notification frame 1157 target 7003 types may include: play another video, seek

to a different location in the same video, seek to an ASF Marker defined in the current video, display text on top of the video, or navigate to a location at a URL whose corresponding data is displayed in selected HTML frame. For each target type, the ENSFrameLinkData class includes a further set of parameters, or descriptors, used to further identify the target 7003. For example, if a target 7003 type navigate to a location at a URL is selected, then the URL Boolean variable is set to true and the relevant descriptors, including the URL name, location and selected frame, are provided. The ENSHotspot class includes parameters to define hotspot 8003 geometry and cursors, in addition to the target 7003 parameters inherited from ENSFrameLinkData.

Hypervideo data buffers will now be further illustrated by exemplifying their use. A hypervideo data buffer may be created for a frame for which an elliptically shaped hotspot is defined. The elliptically shaped hotspot may be linked to a target 7003 which is of the type that displays text on top of the video. Thus, during hypervideo performance, upon actuating (e.g. clicking on) the elliptically shaped hotspot, text may appear over the video. The frame may also be a notification frame 1157 whose target 7003 is the type that navigates to a location at a URL (e.g. for a web site) whose corresponding data is displayed in a selected HTML frame. The hotspot 8003 center is located at coordinates (100,100) and the hotspot's horizontal and vertical radiuses are respectively (20,60).

To implement the elliptically shaped hotspot and notification frame 1157, an ENSFrameInfo object is instantiated. An ENSHotspot object is also instantiated. The ENSHotspot object may include an ENSEllipse object having a text type target. An ENSEvent object is created having a target 7003 type that navigates to a location at a URL whose corresponding data is displayed in a selected HTML frame. Both ENSHotspot and ENSEvent objects are added to the ENSFrameInfo object. An ENSBuffer object is then instantiated. The ENSFrameInfo object is serialized into the ENSBuffer object.

The resulting hypervideo data buffer has the following structure:

Byte 0: t (stands for true, this is a notification frame)

---- the notification frame description ------

Byte 1: t (true, this notification frame has a URL type target)

Byte 2: 22 (the length of the URL's name)

Bytes 2-20: http://www.website.com (the URL's name)

Byte 21: f (false, this is not a seek to time target)

Byte 22: f (false, this is not a seek to marker target)

Byte 23: f (false, this is not a text target)

--- the hotspot description ------

Byte 24: f (false, this is not a URL target)

Byte 25: f (false, this is not a seek to time target)

Byte 26: f (false, this is not a seek to marker target)

Byte 27: t (true, this is a text target)

Byte 28: 8 (the text's length)

Bytes 29-36: the text

Bytes 37: 11 (hotspot's description length)

Bytes 38-48: Description

Byte 49: 1 (the number of the hotspot click cursor)

Byte 50: 1 (the number of the hotspot over cursor)

Byte 51: 100 (the ellipse center x-axis coordinate)

Byte 52: 100 (the ellipse center y-axis coordinate)

Byte 53: 20 (the ellipse horizontal radius)

Byte 54: 60 (the ellipse vertical radius)

Embedding the Buffers inside an ASF File

This section describes how hypervideo data buffers may be placed in a streaming media file, such as an ASF version 1 file, which may include audio and video streams. As previously stated, the hypervideo data buffers may be inserted

into any media file that maintains a private data stream. In one embodiment, the hypervideo data buffers may be inserted into the script command stream in an ASF version 1 file. Script commands, used in the script command stream, have a type field, an argument field and an execution time field. In another embodiment, each hypervideo data buffer may be associated with a script command of type 'VANS'. In another embodiment, the hypervideo data buffers may be streamed across a network 1650 from a server to a client including a Netshow (TM) player. The Netshow (TM) player may fire a script command, and hence the corresponding feature defined in the argument field, at the corresponding execution time. When a script command is fired, a corresponding media object data resident in the Netshow (TM) player may be modified.

Additional Techniques to Reduce Network Capacity Consumption by a Hypervideo Data Stream

The features of the techniques described above may be used to implement yet another technique for streaming hypervideo data in a manner that consumes less network capacity. For example, certain interactive hotspot 8003 attributes, such as click and over cursors, marker 1155 and target(s) 7003, may be more likely to remain constant throughout the life-time of the hotspot 8003. Specifying such attributes more than once for each hotspot 8003 (*e.g.* for each frame in which the hotspot 8003 occurs) may be undesirably redundant because such a technique consumes relatively large network capacity. Therefore, these static hotspot features may be defined in the header 1402 of the hypervideo data stream. On the other hand, however, providing such hotspot attributes, for example, for each video frame may be desirable because it may more readily permit streaming hypervideo to be randomly accessed, in mid-stream.

Further, the content of successive video frames may be relatively time invariant (e.g. change little from frame to frame). In such cases, encoder-decoders (CODECs) may be used to compress video data to reduce the bandwidth of the video stream. Further, in such cases, media object geometry may also be compressed, for example with CODECs, to reduce the bandwidth of the hypervideo data stream 1660. In one embodiment, the CODECs may

compress the hypervideo data stream 1660 because the hypervideo data may be relatively time invariant also. In another embodiment, a CODEC may define a key frame in which media object information, including geometry, is encoded. Subsequent frames may be encoded based upon their differences, for example in hotspot 8003 geometry, with respect to the key frame. In another embodiment, which is relatively lossy, media object geometry, in subsequent frames may be defined by a vector, such as a motion vector.

In another embodiment, redundant hypervideo data may be stored in a header 1402 of the hypervideo data stream 1660 that may be streamed to a run time module 1101 in a client. In one embodiment, the redundant hypervideo data may be referenced by unique identifiers, or pointers, 1406 in the body 1404 of the hypervideo data stream 1660.

The header 1402 may be stored in a buffer memory of sufficient size, for example in a client of a client-server network, during the display of the corresponding hypervideo 10,007. Further, because they may be loaded into the buffer memory, relatively large headers 1402 may increase latency, or download time, of the hypervideo data stream 1660. Latency may be more problematic for multimedia files that are randomly accessed after their beginning. In one embodiment, regardless of access location, a complete header 1402 may have to be loaded into the buffer memory before multimedia performance may begin.

Hypervideo Data Streaming with Common Data Pointers

One embodiment of a technique of using pointers 1406, which may be referred to as the common data pointers method, will now be described. In this embodiment, typically no redundant, or common, media object data may be encoded in the hypervideo data stream 1660. The encoded stream may include only time-dependent data, such as media object geometry and shape. The common media object attributes, which are typically time invariant, and thus typically need only to be encoded once in the hypervideo data stream 1660, may be stored in the header 1402 of the hypervideo data stream 1660. The common media object attributes are coupled to the media objects by pointers 1406. In one embodiment, the pointers 1406 are media object identifiers (IDs).

One embodiment of a hypervideo data stream 1660 will now be illustrated, and includes a description of the encoded parameters within the header 1402 and body 1404.

Header:

The header 1402 of the hypervideo data stream 1660 may include common attribute data associated with media objects encoded in the hypervideo data stream 1660. In one embodiment, the header 1402 may include the following common attribute parameters (primarily related to interactive features of hotspots 8003):

Media Object ID: A unique identifier of a media object 8003.

Cursors: User-defined cursors, including both the over and click cursors.

Marking: Marker 1155 definition.

Targets: Target 7003, or link, definition.

Horizontal and Vertical Resolution, and Time Units, as described below.

Dimensions of the Media Window in which the Media was Authored.

The header 1402 will now be described in further detail. In one embodiment, the field for the Hotspot ID may be 32 bits in length. The cursor fields will now be described.

In another embodiment, the field for the over cursor may be 2*8 bits. If the first byte is:

0x0, then the over cursor is not changed from the click cursor, and thus is the same as the click cursor;

0x1 to 0xfe, then corresponding predefined cursor shapes are used; and

0xff, then the next byte holds a pointer to a location in a cursor table associated with a cursor shape to be displayed.

In another embodiment, the field for the click cursor is 2*8 bits. The click cursor is defined in a manner similar to the definition of the over cursor.

In yet another embodiment, the field for the marker 1155 is 4*8 bits. The type of marker 1155 is described in 8 bits. If bit 1 is set to one, then a flag is set to indicate a blinking marker. If bit 2 is set to one, then the marker 1155 is pointer sensitive, appearing when the pointer is positioned over the region corresponding to the media object. These techniques are further illustrated below.

Body:

In one embodiment, the body 1404 may include the following parameters for each media object:

Media Object ID: Uniquely associated with a media object.

Media Object Shape: Including Rectangle, Ellipse, Triangle, Polygon, Circle, etc.

Media Object Geometry: Includes the size and position of the media object. The media object geometry may be encoded using relative coordinates with respect to the origin of the playback window to facilitate image map resizing to permit media object dimensions to be resized proportionally with the playback window. For media objects having primitive shapes, bounding rectangle coordinates may be provided, and include sufficient information to determine the geometry of the media object. Bounding rectangles, which are known to persons skilled in the art, are rectangles whose sides contact the periphery of a primitive shape, but do not intersect with the interior of the primitive shape. Bounding rectangles may be used to describe a triangle that is an isosceles triangle, for example, with a horizontal base. Other types of triangles, however, may be defined with a polygon shape.

In one embodiment, the field for the media object shape has 8 bits. The data in this field identifies the shape of the media object (e.g. rectangle, triangle, ellipse, circle, polygon, etc.). For primitive shapes (e.g. rectangle, triangle, circle and ellipse), the media object dimensions are defined by a bounding rectangle. In one embodiment, the bounding rectangle coordinates, used to define a primitive shape media object, may be stored in a field of 4*16 bits. In another embodiment, only the top-left and bottom-right coordinates of the bounding

rectangle need be stored in this field. In one embodiment, for polygon shapes, the number of vertices are defined in a 16 bit field. The vertices' coordinates are defined in a field whose size equals the number of vertices 2 *16 bits.

In one embodiment, a common data parameter in the header 1402 may be associated with a media object parameter in the body 1404 by using an identifier. In another embodiment, the common data parameter for a media object may be used in the body 1404 by inserting the identifier (e.g. media object ID) corresponding to the common data parameter in a field, in the body 1404, associated with the media object parameter.

Scalable Streaming

In one embodiment, latency and bandwidth of a hypervideo data stream 1660 can be diminished by streaming the hypervideo data in a scalable manner. The bandwidth of the hypervideo data stream 1660 may be varied to accommodate the bandwidth available in the system over which the hypervideo data stream 1660 is transmitted. In this embodiment, substantial amounts of media object data may be encoded to form a relatively wide bandwidth hypervideo data stream. Alternatively, solely basic hypervideo data may be encoded to provide a relatively narrow bandwidth hypervideo data stream. One embodiment of a scalable hypervideo data stream will now be described.

One embodiment of a scalable hypervideo data stream includes three layers of hypervideo data. In one embodiment, different layers of hypervideo data stream 1660 describing one or more media objects are coordinated, utilizing object ID and time (or frame) parameters, in a manner understood by one skilled in the art. In another embodiment, the first layer of the hypervideo data stream 1660 includes only basic hypervideo data, such as media object geometry, and no interactive data. The basic hypervideo data is intended to be utilized by any hypervideo application, such as one that utilizes indexing of media based upon media objects. A hypervideo data stream 1660 including only the basic hypervideo data (e.g. the first layer) consumes relatively little network capacity.

In another embodiment, a second layer includes basic interactive features for the media object, including marker 1155, cursor and target 7003 data. The

basic interactive features may be selected from predefined sets of markers 1155 (e.g. NoMarking, Highlight, Dark, Negative, Gray, RGB, etc.), cursors (e.g. NoCursor, HandCursor, etc.) and targets 7003 (e.g. URL flip [which changes the URL in a frame of browser], Seek - a time code or frame, Switch stream, Pause, Exit, etc.).

In another embodiment, the second layer may also provide other interactive features, such as markers 1155 that alternate, including a blinking marker. The blinking marker is a marker that is turned on and off repetitively. The second layer may also enable the marker 1155, cursors, and target(s) 7003 of a media object to change at different times or frames, for example, of a video in the multimedia stream. For example, a first instance of a media object may have a first marker, a first set of cursors, and a first target, and may be associated with a first time period, or set of frames, in a video. A second instance of the same media object may have a second marker, a second set of cursors, and a second target, and may be associated with a second time period, or set of frames, in the video. However, hypervideo data for a media object may be relatively time invariant, diminishing the number of instances of a media object, thus providing relatively high data compression rates and relatively low network capacity consumption.

The third layer may include enhanced hypervideo properties, enabling user-defined markers, cursors and targets for hotspots 8003 to be defined in a hypervideo data stream 1660. Thus, such enhanced hypervideo properties may be used, without requiring customized software for implementing the enhancements, with the authoring tool 1001 and run-time module 1001. As a result, a user does not need to provide external code to utilize the enhanced hypervideo properties.

Each of the three layers may provide a different scope of media object definition, including for geometry and interactivity. However, in other embodiments, the layers may be reorganized, for example, by merging or sub-dividing the layers. In a further embodiment, for example, the first and second layers may be combined into a single layer to provide a non-scalable stream, for example, for a hypervideo system that does not support scalable streams. In one

embodiment, the single layer may include media object geometry, and cursor, marker 1155, and target 7003 definition from the predefined sets discussed herein.

An exemplary embodiment of the layers is described below:

First layer:

Media Object Shape: Rectangle, Ellipse, Triangle, Polygon, etc.

Media Object Geometry: Indicates the size and position of the media object. For primitive shapes, two sets of coordinates of bounding rectangle may only be provided. The media object geometry may be encoded using relative coordinates to facilitate image map re-sizing, as further described herein.

Second Layer:

Cursors: Permits each media object to be assigned at least two cursors, the over and click cursors. The over cursor may be displayed, instead of the pointer, when the pointer is positioned over the region corresponding to the media object. The click cursor may be displayed instead of either the pointer or over cursor, when the media object has been actuated, for example by a mouse "click" when the pointer is over the region corresponding to the media object. In one embodiment, in the second layer, only cursors, such as NoCursor or the HandCursor, from a pre-defined set may be assigned to a media object.

Marker: Permits each media object to be visually marked by a marker 1155. In one embodiment, in the second layer, only markers 1155, such as NoMarking, Highlight, Dark, Negative, Gray and RGB, from a pre-defined set may be assigned to a media object.

Target (Commands): Permits each media object to be coupled to target(s) 7003. In one embodiment, in the second layer, only target(s), such as URL Flip, Seek, New Media, Pause, Resume and Exit, from a pre-defined set may be assigned to a media object.

Third layer:

Cursors: Permits user-defined cursors to be assigned to a media object by encoding the cursor's path in the hypervideo data stream 1660.

Marking: Permits a user-defined marker, such as an overlay image, or any low-bandwidth media, to be assigned to a media object by encoding the marker's 1155 path and its display parameters (key color, relative position, etc.) in the hypervideo data stream 1660.

Target(s) (Commands): Permits linking target(s) 7003, which may be any media, to a media object by specifying the target(s)'s location. Additionally, target display parameters, such as position, size, and duration, for example of a target window, may be provided in the third layer.

In one embodiment, if a media object parameters, such as cursor, marker 1155 or target(s) 7003, are defined, for example contemporaneously, in two layers (e.g. the second and third layers), the parameter definitions in the higher layer (e.g. third layer) may be used. If one or more parameters are not encoded in the higher layer (e.g. third layer), the parameters defined in the lower layer (e.g. second layer) may be used.

As mentioned above, a hypervideo data stream 1660 renderer may provide an extensive API to facilitate retrieval of streamed hypervideo data, such as media object information. For example, media object geometry may be extracted with the API, for example when only the first layer is streamed. Extended, or other, features may be implemented through the mentioned API.

Scalable Hypervideo Data Stream Format

Another embodiment of the hypervideo data stream 1660 will now be described in further detail.

Header:

The header 1402 includes, in addition to other media parameters, global parameters required in the hypervideo data stream 1660.

Field Name	Field Type	Size (bits)	Description
Horizontal Resolution	UINT	16	The horizontal resolution of media objects. This parameter is used to scale the media object's geometry parameters, as described above.
Vertical Resolution	UINT	16	The horizontal resolution of media objects. This parameter is used to scale the media object's geometry parameters, as described above.
Time Units	OBTimeUn	8	The time resolution, or units, of the visual media. This parameter defines the units used for time stamps.

The Horizontal and Vertical Resolution parameters determine the resolution, or units, of media objects's geometry parameters. The Horizontal and Vertical Resolution parameters may be used to scale media objects's geometry parameters. The Horizontal and Vertical Resolution parameters recite a number of 'logical units' for frame width and height. This relative representation readily facilitates re-sizing media objects and media scaling, or in other words allows media object size to be varied with the size of window in which the media object appears.

The Time Units parameter determines the time units with which the media object's time parameters, including start time, are defined. In one embodiment, the time units are defined in milliseconds (MS). In another embodiment, the time units are defined by frame numbers.

One embodiment of the body 1404 of the hypervideo data stream 1660 will now be described. This embodiment is comprised of multiple layers. The structure of each layer, including the media object data therein encoded, or defined, will be described. The encoded media object data may include media object geometry and interactive features, as illustrated above.

A media object may be defined for a finite period, or duration, of frames or time. In one embodiment, the finite period is valid from the Start Time of first data defining a first instance of the media object through the Start Time of second data defining a second instance of the media object. In another embodiment, the media object is terminated by issuing dummy data for the media object, including the media object's unique ID and End Time, which respectively specify the media object to be terminated, and the time to terminate the media object.

In another embodiment, Time Stamps are provided, and can be used to define the beginning of a one set of media objects, and the end of another set of media objects. All media objects instantiated following the issuance of a time stamp are valid until the issuance of another Time Stamp. In this embodiment, all media objects share a common, and typically relatively fine, time resolution defined by the Time Unit parameter.

Because this technique may use relatively finer time resolution than other techniques, this technique may be relatively inefficient for encoding media objects having relatively constant geometry for relatively long time periods. Media objects having relatively constant geometry for relatively long time periods must be repeatedly encoded with this technique. As a result, the hypervideo data stream 1660 bandwidth may be relatively large compared to other techniques. In the previously described technique, media objects can be encoded with a relatively wide time resolution.

One embodiment of the data structures of the first, second and third layers is described below.

First Layer

Field Name	Field Type	Size (bits)	Description
Media Object ID	UINT	16	A unique identifier of the media object.
Start Time	UINT	32	The starting time of the instance of the media object.
Media Object Shape	OBShape	4	The shape of the instance of the media object.
Media Object Flags	OBFlags	4	Different flags assigned to the media object that may be used by an external application.
Media Object Geometry		(4*16) or (N*2*16)	
For primitive shape media objects (such as a Rectangle, Triangle, Ellipse, Circle, etc.)			
Left	UINT	16	X coordinate of the top left corner of the bounding rectangle.
Top	UINT	16	Y coordinate of the top left corner of the bounding rectangle.
Right	UINT	16	X coordinate of the bottom right corner of the bounding rectangle.
Bottom	UINT	16	Y coordinate of the bottom right corner of the bounding rectangle.
For Polygon shape media objects			
X1	UINT	16	X coordinator of the first vertex of the polygon.
Y1	UINT	16	Y coordinate of the first vertex of the polygon.
Xn...	UINT	16	X coordinate of the n-th vertex of the polygon.
Yn...	UINT	16	Y coordinate of the n-th vertex of the polygon.

Field Name	Field Type	Size (bits)	Description
XN	UINT	16	X coordinate of the last vertex of the polygon.
YN	UINT	16	Y coordinate of the last vertex of the polygon.

The Media Object ID is a unique identifier of a media object throughout the life span of the media object. The media object ID is also used to reference, or associate, the media object extended parameters in higher (e.g. second and third) layers, with the geometrical parameters in the first layer.

The Start Time parameter uses the units defined in the Time Units parameter in the header 1402. The Media Object Shape parameter is used to select one of a group of pre-defined shapes, including Rectangle, Triangle, Ellipse, Circle, Polygon, etc. The Media Object Geometry parameters are used to define the dimensions and position of the media object using the Horizontal/Vertical Resolution units defined in the header 1402. Primitive shapes, such as a Rectangle, Ellipse, Circle, Triangle, etc., are defined by a bounding rectangle.

Second Layer

Field Name	Field Type	Size (bits)	Description
Media Object ID	UINT	16	A unique identifier of the media object.
Start time	UINT	32	The starting time of the instance of the media object.
Over Cursor	OBCursor	4	The shape of the cursor when moving over the media object.
Click Cursor	OBCursor	4	The shape of the cursor when clicking on the media object.
Marking		8	The visual properties of the media object.

Field Name	Field Type	Size (bits)	Description
Marking Type	OBMark	6	The predefined marking to use.
Blink	Boolean	1	Determines whether marker blinks.
Mouse Sensitive	Boolean	1	Determines whether marker is mouse sensitive.
Link type	OBLinkType	8	The command type which will be activated when actuating the object.
Link Command:			
For URLCommand			
URL	string	N	The full URL address.
Location	string	N	The location in the URL page to flip to.
Frame	string	N	The frame within the browser frame-set to flip the URL in.
For Seek Commands			
Time/Frame/Marker	MS/UNIT/ MARKER	32(64)/32/ 32	The point in the stream to seek to.
For New Media Commands			
Media Path	string	N	The full path of the new media to be played.

The media object ID is a unique identifier of the media object, throughout its life span. The media object ID is also used as a reference to the media object parameters in the higher (e.g. third) layers, and to the geometrical media object parameters in the first layer.

The Start Time parameter is interpreted according to the Time Units parameter in the hypervideo data stream 1660 header. OBCursor defines one of a set of pre-defined cursors (e.g. NoCursor, Arrow, Cross, Hand, Back, etc.). OBMark defines one of a set of pre-defined markers 1155 (e.g. NoMarking,

Highlight, Negative, Dark, Gray, RGB Meshes, RGB Filters, some pre-defined images marking such as an arrow or a flag pointing to the hotspot, etc.).

If the Blink flag is on, or set to one, the selected marker will blink at a constant rate. If the Mouse Sensitive flag is on, or set to one, the selected marker will be displayed, for example in lieu of a pointer, only when the pointer is positioned over the media object.

The Link Type defines the target 7003 linked to the media object. OBLinkType defines one of a set of pre-defined command (e.g. URL (flip a URL page), NewMedia (start playing a new media instead of the present media), SeekToFrame, SeekToTime, SeekToMarker, Pause, Resume and Exit).

Third Layer

Field Name	**Field Type**	**Size (bits)**	**Description**
Media Object ID	UINT	16	A unique identifier of a media object.
Start Time	UINT	32	The starting time of the media object.
Over Cursor	string	N	The full path of a cursor file. The over cursor will define the shape of the cursor displayed when the pointer is moved over the media object.
Click Cursor	string	N	The full path of the cursor file. This cursor will define the shape of the cursor when actuating, such as by clicking, the media object.
Marking Media	string	N	The full path of the media used to mark the media object.
Marking Attributes		8	
Horizontal Anchor	OBMrHAnc	2	Horizontal axis position of media object anchor.
Vertical Anchor	OBMrVAnc	2	Vertical axis position of media object anchor.

Field Name	Field Type	Size (bits)	Description
Transparency	Boolean	1	Use an opaque or transparent overlay.
Blink	Boolean	1	Determines whether marker 1155 blinks.
Mouse sensitive	Boolean	1	Determines whether marker 1155 is be mouse sensitive.
Not used		1	
Marking Media Horizontal Offset	UINT	16	The horizontal offset of the marker 1155 from the media object anchor.
Marking Media Vertical Offset	UINT	16	The vertical offset of the marker 1155 from the media object anchor.
Marker Transparency RGB Color (only if the transparency flag is set or for text media).	RGB	24	The background color of the marking media.
Marking Media Blink Rate (only if the Blink Flag is set)	UINT	16	The rate at which the marking blinks (in Time Units as defined in the stream header).
The following Link Media, or Target, parameters can be repeated more than once for multiple Link Media to be support.			
Target Media	string	N	The full path of the media which is linked as target 7003 to the media object.
Target Media Sequence Attributes		16	
Link Media Level	UINT	8	Level in which media object is located. The level may establish the order in which multiple targets are executed, as previously described above.
Leader	Boolean	1	Flag to indicate whether the target is a leader in a layer.

Field Name	Field Type	Size (bits)	Description
Pause/ Close/ Keep Playing base media		2	What to do with the base media when the link media starts to play.
Pause/ Close/ Loop Target		2	What to do with the target when it ends.
Not used		3	
Target Display Attributes		(4*16)	
Left	UINT	16	X coordinate of the top left corner of the bounding rectangle.
Top	UINT	16	Y coordinate of the top left corner of the bounding rectangle.
Right	UINT	16	X coordinate of the bottom right corner of the bounding rectangle.
Bottom	UINT	16	Y coordinate of the bottom right corner of the bounding rectangle.

Any of the string parameters may contain a NULL if the parameters are not applicable, or if parameters in a lower layer parameters are to be used instead. If the Marker Media or the Target Media path string is set to NULL, all other corresponding parameters will not be encoded when authored or recognized when played.

The Marker Media file may be any of the following types: Image file, Animation file, Low Bandwidth Video file or Text file. These media types can all be used to mark a media object by overlaying the media relative to corresponding media object position. Note, the transparency attributes are defined by the transparency flag.

To position the marker media relative to the hotspot 8003, an anchor is defined. In one embodiment, the anchor is a position in the hotspot, such as left, center, right or cursor position for the horizontal dimension, and the top, center,

bottom or cursor position for the vertical dimension. The exact position of the overlaying marker can be described by defining the hotspot anchor and the media offset from this hotspot anchor. If the marker media is text media, and the transparency flag is set to off, the transparency color will be used as a background color for the text media.

The Target path can point to any file type, such as a media file, that is supported by a recipient client. In one embodiment, an relatively unlimited number of targets 7003 may be associated with each instance of a media object. The target(s) 7003 defined in the third layer supplement the target(s) 7003 defined in the second layer..

The Target Media Sequence attributes define the sequence in which multiple targets 7003 are executed. Each target 7003 is assigned to a level 7601. Targets on the same level, play simultaneously. Targets, on a layer such as the second and successive levels 7601, begin playing when the preceding level 7601 is terminated, and the target(s) 7003 on the predecessor level 7601 have stopped playing. a level 7601 terminates when the target 7003 defined as the leader target 7602 ends playing, as described above. The Target Media Sequence attributes also determine how base target(s) 11,001 are manipulated when the media object is actuated, and when each target 7003 ceases playing. The Target Media Display attributes define the position and size of the window in which a target 7003 is displayed upon its execution.

2.4 Streaming Hypervideo Authoring Tool

Streaming hypervideo can be authored and displayed by modified versions of the authoring tool 1001 and run-time module 1101. This section describes modified versions of authoring tools 1001 and run-time module 1101 for two streaming formats, Netshow (TM), by Microsoft (Redmond, WA), and RealVideo (TM), by Real Networks (Seattle, WA). The modified versions of the authoring tools 1001 and run-time module 1101 are respectively derived from the authoring tool and run-time module 1101 described above.

A modified run-time modules 1101 plays streaming media from Netshow (TM). a Real Networks (TM) player plays streaming media from RealVideo

(TM) video servers. Netshow (TM) uses ASF. RealVideo (TM) uses a different video streaming format, the RM format.

2.41 Hypervideo Authoring Tool for Netshow (TM)

Modified versions of the hypervideo authoring tool 1001 and run-time module 1101 for Netshow (TM) will now be described. The modified version of the authoring tool 1001 for Netshow (TM) can create Netshow (TM) compatible hypervideos.

The modified version of the run-time module 1101 for Netshow (TM) can display the Netshow (TM) compatible hypervideos created by the modified version of the authoring tool 1001 for Netshow (TM). In one embodiment, the modified version of the run-time module 1101 for Netshow (TM) includes an ActiveX control, which facilitates embedding hypervideo in HTML pages and other applications supporting OLE/ActiveX controls. The ActiveX control in the modified version of the run-time module 1101 for Netshow (TM) may make use of the Netshow (TM) ActiveX control to stream video.

Modified Version of Authoring Tool for Netshow (TM)

The modified version of the authoring tool for Netshow (TM) may include the following functions. One or more .AVI and .MOV files may be imported into the modified version of the authoring tool for Netshow (TM) which may be used to create a hypervideo 10,007. In other embodiments, other types of media files, such as motion picture experts group (MPEG) files, may be used to create a hypervideo 10,007. Media objects can be defined and tracked, automatically or manually, in the visual media (e.g. the AVI and .MOV files). Media object properties, such as name, media object description, click and over cursors, and linked targets 7003 (e.g. URL, seek to frame, play another ASF file), may be defined for each media object. ASF markers (not to be confused with media object, or hotspot, markers) may be used to identify specified times or frames.

Time-based events, including ASF script commands, (e.g. description, URL flip, play another ASF file) may be defined. Properties for an ASF file (e.g.

author, copyright, description, rating, title, bit rate, and window size) may be defined. Hypervideo data may be saved in a project file 1670. Video, and other multimedia data (e.g. audio and hypervideo data), may be exported into an ASF file.

GUI Design

The modified version of authoring tool for Netshow (TM) 15,000 has a GUI 15,001, illustrated in Figure 15, that is similar to the GUI 10,001 for the authoring tool 1001 described above. In one embodiment, the GUI 15,001 of the modified version of the authoring tool for Netshow (TM) 15,000 may include the following GUI 15,001 components: menu bar 15,005, main tool bar 15,007, Media Warehouse window 15,003, Workshop Window 15,011 and two floating tool bars, the hotspot tool bar 15,008 and the workshop tool bar 15,009.

The menu bar 15,005 may include the following headings and commands:

File:

New

Open

Save

Save As ...

Revert

Send ...

Import Media File

Most Recently Used (MRU) List

Exit

Edit:

Undo

Cut

Copy

Paste

Delete

Select All

Media:

Large Icons

Small Icons

Report View

Sort By ...

Export ASF

Workshop:

Play

Pause

Mark IN Frame

Mark OUT Frame

Select

Define New Hotspot ...

Auto Track

Options:

Settings ...

Window:

Workshop Tool

Hotspot Tool

Close

Close All

Help:

Index

Using Help

About Authoring Tool (V-Active (TM))

The main tool bar 15,007 includes buttons, which may be associated with menu bar 10,005 commands, and include: New, Open, Save, Revert, Undo, Cut,

Copy, Paste, Settings, About. The functions of the foregoing commands and buttons are known and understood by those skilled in the art.

In one embodiment, the Media Warehouse window 15,003 may support media elements 1690 whose corresponding media files 1680 are either in AVI or QuickTime formats. The Import Media File dialog 2001, thus, may only present *.avi and *.mov file selection. In other embodiments, the Import Media File dialog 2001 may display other types of media files, including MPEG video files.

The Media Warehouse window 15,003 control bar will include new Export and Export All buttons. In one embodiment, the Relative Path Name button may be included in the Media window control bar. By actuating a switch, such as the right button of a mouse, while the pointer is over a thumbnail illustration 4007 in the Media Warehouse window 15,003, various commands, in a displayed menu, are displayed and can be executed. One command facilitates the display of a list of all the hotspots 8003 defined in the media corresponding to the thumbnail illustration 4007. This command may also permit the target(s) 7003 of each hotspot 8003 to be illustrated. Other commands that can be actuated include: Export, Export as..., etc..

Upon placing the pointer on a thumbnail illustration 4007 in the Media Warehouse window 15,003, and doubling actuating (e.g. clicking), a switch (e.g. lefthand mouse switch), the Video Media Properties dialog will be displayed. The Video Media Properties dialog includes the General, Preview, Hotspots, Events, Markers and ASF properties sheets.

In one embodiment, the Workshop window 8001 of the modified version of the authoring tool for Netshow (TM) 15,000 may include the features of the Workshop window 8001 described above, with the exception that:

(1) the Preview button 2003 may be omitted;

(2) the technique for linking target(s) 7003 to a hotspot 8003 may differ. In one embodiment, targets 8003 may only be defined in the Hotspot Properties sheet 7101. In this embodiment, targets may no longer be defined using a drag and drop technique with media elements 1690 in the Media Warehouse window 15,003;

(3) an indication of an ASF marker with its label may be displayed on top of the current working frame; and

(4) ASF markers may be edited, removed and added by accessing the Media Warehouse window 15,003 only.

In another embodiment, the Set Target button 9015 may be present in the Workshop Tool bar to permit a target 7003 to be defined for a media object.

Other user interfaces, including property sheets, of the GUI 15,001 of the modified version of the authoring tool for Netshow (TM) 15,000 will now be described.

Hotspot Properties Sheet

The GUI 15,001 of the modified version of the authoring tool for Netshow (TM) 15,000 may include a Hotspot Properties Sheet 16,101, as illustrated in Figure 16. Each Hotspot Properties Sheet 16,101 is uniquely associated with a hotspot 8003. The Hotspot Properties Sheet 16,101 may include a Targets, or link, page 16,001 that facilitates linking target(s) 7003 to the hotspot 8003.

The Targets page 16,001 may permit the following types of targets 7003 to be linked to a hotspot 8003:

a new ASF stream 16,201 to be played by the Netshow ActiveX (TM) control;

flip URL 16,203; and

seek to another position in video by frame number 16,205 or ASF marker 16,207.

In one embodiment, a target 7003 may be browsed in a frame screen 16,901 on the Targets Page 16,101. Specific frames or time base times in the target 7003 may be displayed by specifying a corresponding ASF marker. The frame screen 16,901 replaces the frame window 7901 used in the GUI 10,001 of the authoring tool 1001 described above.

The General Properties page of the Hotspot Properties Sheet 16,101 may be similar to the General Properties page 7103 of the GUI 10,001 of the authoring tool 1001 described above.

Media Properties Sheet

The GUI 15,001 of the modified version of the authoring tool for Netshow (TM) 15,000 may include a Media Properties Sheet 17,003, illustrated in Figure 17. The Media Properties Sheet 17,003 may be similar to the Media Properties Sheet 5003 of the GUI 10,001 of the authoring tool 1001 described above.

In one embodiment, the Media Properties Sheet 17,003 may be modified to define time-based events, markers 1155 and ASF file properties by including additional pages such as events, properties and markers pages.

The Events page 17,101, illustrated in Figure 17, facilitates implementation of time-based events (e.g. ASF Script Commands). On the Events page 17,101, a frame, or time code, 17,103 and event 17,105 to be activated upon display of the frame 17,103 may be specified. The events that can be specified include change the ActiveX control's description 17,111, URL flip 17,113, and launch a new ASF file 17,115. This technique is similar to notification frames 1157 described above.

The ASF Markers Page facilitates the assignment of ASF markers for specific frames, or times in a time base. The ASF markers can also be viewed in the Workshop window 15,011.

The Media Properties Sheet 17,003 may include a Properties page 18,001, illustrated in Figure 18. The Properties page 18,001 may be similar to the Details page 5001 of the GUI 10,001 of the authoring tool 1001 described above. The Properties page 18,001 facilitates definition of properties for ASF media files. The properties may include title 18,003, author 18,005, copyright 18,007, description 18,009 and rating 18,011.

Exporting Media

Hypervideo data and media may be exported by the authoring tool 1001 into an exported file. For example, the modified version of the authoring tool for Netshow (TM) 15,000 may export the hypervideo data to an ASF file so that the ASF file includes both video and hypervideo data. In one embodiment, upon creating an exported file, a Properties page, for example for the ASF file properties, is displayed. The Properties dialog may be similar to the Properties

dialog used in the Microsoft Netshow Editor. The properties dialog may permit specification of bit rate properties and modification of ASF properties.

In another embodiment, if the exported file is saved, a Save file dialog may appear. In a further embodiment, an ASF file including video data and hypervideo data, including other authored data, e.g. markers, script commands and hotspots, is saved

ActiveX (TM) Control of Modified Version of the Run-Time Module for Netshow (TM)

The modified version of the run-time module for Netshow (TM) has an ActiveX (TM) control wrapped around the Microsoft (TM) Netshow (TM) On-Demand Player ActiveX (TM) control. As a result, hypervideos 10,007 in ASF can be displayed by the On-Demand Player ActiveX (TM) control. The ActiveX (TM) control of the modified version of the run-time module for Netshow (TM) may have the following functions:

Open, play, stop and pause ASF files, including those ASF files created by the modified version of the authoring tool for Netshow (TM) 15,000;

Signal the existence of a hotspot 8003 by changing the mouse pointer (e.g. over cursor) when the pointer is positioned over the hotspot 8003;

Execute target(s) 7003 in response to the activation of a hotspot 8003. The target(s) 7003 may include: flip URL, seek to a position (e.g. frame) of a video, and play another ASF file;

Activate time-based events (e.g. with ASF Script commands) including: change status bar description, flip URL, and play another ASF file; and Display properties of an ASF file.

ActiveX (TM) Control GUI for Modified Version of the Run-Time Module for Netshow (TM)

In one embodiment, the ActiveX (TM) control for the modified version of the run-time module for Netshow (TM) may be an embedded control. The Active X (TM) Control for the modified version of the run-time module for Netshow (TM) GUI 19,001, illustrated in Figure 19, may include one or more of

the following bars: a status bar 19,013, a display bar 19,011, and a control bar 19,003.

The status bar 19,013 may be used to display text, describing the status of the ASF streaming process. The status bar 19,013 may be changed using the change description event of ASF. The display bar 19,011 may indicate the current playing time of a video. The display bar 19,011 may be shown only in accordance with the ShowDisplay property. The control bar 19,003 may include a Play/Pause button 19,005, a Stop button 19,007 and a Time Slider 19,009. The ActiveX (TM) control for the modified version of the run-time module for Netshow (TM) GUII 19,001 may also have a properties dialog.

When a pointer is over the Active X (TM) Control for the modified version of the run-time module for Netshow (TM) GUI 19,001, a pop-up floating menu may be displayed when a button, such as the right mouse button, is actuated, or clicked. The menu may include the following selections: Play, Pause, Stop; Display, Controls, Status bar; Properties; About. The Properties dialog box may resemble the Properties dialog box of the Microsoft (TM) Netshow (TM) On-Demand Player, and will allow setting the following properties: play count, auto rewind, auto start, display size, control display, buffering and streaming parameters.

Functional Requirements of the ActiveX (TM) Control

The ActiveX (TM) control for the modified version of the run-time module for Netshow (TM) 19,000 may have the following OLE/ActiveX properties, in addition to the typical properties of ActiveX controls: AutoRewind, AutoStart, FileName, DisplaySize, PlayCount, ShowControls, ShowDisplay, ShowStatusBar, AnimationAtStart, InvokeURLs, TransparentAtStart, TransparentAtStop, StatusDescription. The ActiveX (TM) control for the modified version of the run-time module for Netshow (TM) 19,000 will consume relatively little memory, and will utilize the functionality of the Microsoft (TM) Netshow (TM) On-Demand Player ActiveX (TM) control around which the ActiveX (TM) control for the modified version of the run-time module for Netshow (TM) 19,000 is wrapped.

2.41 Hypervideo Authoring Tool for RealVideo (TM)

A modified version of the authoring tool for RealVideo (TM) will now be described. RealVideo (TM) uses a proprietary format, not the ASF, utilizing Image Maps and Events files. The authoring tool 1001, described above, has been adapted to support the RealVideo (TM) Image Maps files and the RealVideo (TM) Events files, known to persons skilled in the art, to create interactive RealVideo (TM) files. The Image Map files and the Events files may respectively define hotspots 8003 and notification frames 1157, as described above.

In one embodiment, the modified version of the authoring tool for RealVideo (TM) may be designed to operate separately from the RealVideo (TM) run-time environment. In this embodiment, the modified version of the authoring tool for RealVideo (TM) need not be linked to the RealVideo (TM) run-time environment, other than by exporting hypervideo data to the RealVideo (TM) Image Map or Events files whose data is used by the Real Video (TM) run-time environment. Thus, this embodiment may lack the capability to preview a hypervideo.

In other embodiments of the modified version of the authoring tool for RealVideo (TM), other features of the authoring tool 1001 may be removed, or

modified to support the RealVideo (TM) run-time environment. In one embodiment, the modified version of the authoring tool for RealVideo (TM) has:

No project preview;

Limited media importing features;

RealVideo (TM) Image Map export utility;

RealVideo (TM) Events export utility;

Limited target types 7003 corresponding to the RealVideo (TM) link, or target 7003, options; and

No publish utilities for defining the path of the project file 1670 (e.g. .obv file).

GUI Design

One embodiment of the GUI 20,001, illustrated in Figure 20, for the modified version of the authoring tool for RealVideo (TM) 20,000 may be similar to the GUI 15,001 of the modified version of the authoring tool for Netshow (TM) 15,000 described above. However, both GUIs 15,001, 20,001 may not necessarily be identical.

In one embodiment, the GUI 20,001 of the modified version of the authoring tool for RealVideo (TM) 20,000 may include the following GUI 20,001 components: menu bar 20,005, main tool bar 20,007, Media Warehouse window 20,003, Workshop Window 20,011 and two floating tool bars, the hotspot tool bar 20,008 and the workshop tool bar 20,009.

The menu bar 20,005 may include the following headings and commands:

File:

New

Open

Save

Save As ...

Revert

Send ...

Import Media File

MRU List

Exit

Edit:

Undo

Cut

Copy

Paste

Delete

Select All

Media:

Large Icons

Small Icons

Report View

Sort By ...

Export (RealVideo (TM) Image Map) file

Export (RealVideo (TM) Image Map) file As...

Export (RealVideo (TM) Image Map) file All

Workshop:

Mark IN Frame

Mark OUT Frame

Define New Hotspot ...

AutoTrack

Options:

Settings...

Window:

Workshop Tool

Hotspot Tool

Close

Close All

Help:

Index

Using Help

About V-Active

In one embodiment, the main tool bar 20,007 for the modified version of the authoring tool for RealVideo (TM) 20,005 may not include the Project Settings, Project View, Project Preview.

In one embodiment, the modified version of the authoring tool for RealVideo (TM) 20,005 has a Media Warehouse Window 20,003. The Media Warehouse window includes a control bar 20,010. In another embodiment, only AVI & QuickTime media files may be supported by the modified version of the authoring tool for RealVideo (TM). In this embodiment, all other types of media files may not be imported into the Media Warehouse Window 10,003. Thus, the Import Media File dialog 2001 may only present *.avi and *.mov file selection. In this embodiment, other media file types may not be imported even if the "All Files *.*" option is used, or if an .obv file created by the authoring tool 1001 is used. In another embodiment, other media files, such as MPEG files, may be imported.

In one embodiment, the Media window warehouse 10,003 control bar may include 'Export', and 'Export All' buttons. In another embodiment, the Relative Path Name button may be included in the Media window control bar. Upon actuating a switch, such as the right button on a mouse, while the pointer is over a thumbnail illustration 4007 of a media in the Media Warehouse Window 10,003, a menu will be displayed. The menu options then can be selected, and include the commands Export and Export As... . Another menu option that can be selected displays a list of the hotspots 8003 in the media.

In another embodiment, the modified version of the authoring tool for RealVideo (TM) 20,005 has a Media Workshop Window 20,011. Many of the features of the media workshop window 8001 of the authoring tool 1001 may

also be found in the media workshop window 8001 of the modified version of the authoring tool for RealVideo (TM). For example, the menus 15,101, displayed when a switch, such as the right mouse button, is activated regardless as to whether the pointer is over or not over a hotspot, will remain the same. In one embodiment, the authoring tool and player for RealVideo (TM) may support the z-axis ordering of hotspots 8003, as described above. In one embodiment, the Preview button may not be used in the authoring tool for RealVideo (TM) 20,005, for example in the workshop tool bar 20,101.

In yet another embodiment, the modified version of the authoring tool for RealVideo (TM) 20,005 has a workshop tool bar 20,009. Features of the Tools window 9001 of the authoring tool 1001 may also be found in the workshop tool bar 20,009 of the modified version of the authoring tool for RealVideo (TM) 20,000. In one embodiment, however, the Set Target button 9015 may not be included in the workshop tool bar 20,009. In another embodiment of the workshop tool bar 20,009, the Ellipse Button 20,101 is used to draw circles rather than ellipses.

In yet another embodiment, the GUI 20,001 of the modified version of the authoring tool for streaming video, including for RealVideo (TM), may exclude the following features found in the GUI 10,001 for the authoring tool 1001 described above:

Preview window 8101;

Project View 1007;

Project Settings Property sheet;

Pages in the Media Properties Sheet 17,003 except the Properties page 18,001;

Preview sheet in the Settings dialog;

Target Property page in the Hotspot Properties sheet; and

All buttons and controls used to invoke the above features.

Hotspot Properties Sheet

In one embodiment, the GUI 20,001 of the modified version of the authoring tool for RealVideo (TM) may include a Hotspot Properties sheet

21,101, illustrated in Figure 21, similar, but not necessarily identical, to the Hotspot Properties sheet 16,101 described above. Each Hotspot Properties sheet 21,101 is uniquely associated with a hotspot 8003, and may include a general and links, or targets, page. In one embodiment, however, the Hotspot Properties sheet 16,101 may not have a Target Properties page. In another embodiment, the over and click cursors may be dropped from the General page because the RealVideo (TM) system uses only the hand cursor.

Targets 7003 required by the RealVideo (TM) run-time environment may be specified in the Target page 21,001 of the Hotspot Properties sheet 21,101. In one embodiment, the Target page 21,001 may include at least the three types of targets 7003 that are executed upon actuation of a corresponding hotspot 8003, and are supported by RealVideo (TM). The targets 7003 include Seek 21,107, Player 21,103 and URL 21,105.

The Seek 21,107 target identifies a frame number 21,108 in the video currently being displayed. Thus, when a corresponding hotspot 8003 is actuated, the video jumps to the frame having the frame number 21,108, and then that frame and successive frames are then displayed. In one embodiment, the video may be previewed on the Target page 21,001. The frame corresponding to the frame number 21,108 may be selected during such previewing. When the frame is selected, a Mark button 21,111 may be actuated so that the corresponding frame number 21,108 is entered as the Seek 21,107 target.

The Player 21,103 target is another multimedia stream whose location is defined by a stream URL 21,104. Thus, when a corresponding hotspot 8003 is actuated, the multimedia stream at the stream URL is executed, or performed. The URL option, a URL address must be identified.

Media Properties Sheet

The GUI 20,001 of the modified version of the authoring tool for RealVideo(TM) 20,000 may include a Media Properties Sheet similar to the Media Properties Sheet 17,003, illustrated in Figure 17. In one embodiment, the Media Properties sheet may, however, not include Defaults or Publish pages. In

another embodiment, an Events page, such as described above, may be added to the Media Properties Sheet.

Compatibility

In one embodiment, the formats of project files 1670, associated with the authoring tool 1001 and the modified version of the authoring tools for RealVideo (TM) 20,001 and Netshow (TM) 15,001, may be compatible. Project files 1670 (e.g. .obv files) created by one authoring tool may be used by other authoring tools. Each authoring tool may use only the project file 1670 data corresponding to the features that the authoring tool supports.

Exporting Image Map and Event Files

Hypervideo data for RealVideo (TM) multimedia files may include both hotspot 8003 and notification frame 1157 data. Upon authoring a hypervideo 10,007, hotspot 8003 and notification frame 1157 data are initially stored in a project file 1670 (e.g. .obv file). In one embodiment, the project file 1670 will be associated with only one media, e.g. video, file. Subsequently, the modified version of the authoring tool for RealVideo(TM) 20,000 facilitates the exportation of the hotspot 8003 and notification frame 1157 data from the project file 1670 into separate files, respectively an Image Map file and an Event file. The Image Map file stores hotspot 8003 data. The Event file stores notification frame 1157 data. The Image Map and Event files utilize RealVideo (TM) formats known to persons skilled in the art.

In one embodiment, each RealVideo (TM) multimedia file may include only one video file. Thus, in this embodiment, the Image Map and Event files will only contain hotspot 8003 and notification frame 1157 data for this sole video file. The modified version of the authoring tool for RealVideo(TM) 20,000 and associated files will now be further described in view of this embodiment.

When exporting hotspot 8003 or notification frame 1157 data to an Image Map or Event file, the path of the Image Map or Event file must be specified. In one embodiment, Export settings, including default path, can be defined using an Export page, or tab, in the Options|Settings dialog. The Export page can also be

used to define the length of each frame sequence for each Image Map, further discussed below. The length of each frame sequence associated with an Image Map can be determined by corresponding transmission line bandwidth (e.g. 28800 kbps, 33600 kbps, ISDN, etc.), and other parameters set in the authoring tool for RealVideo(TM) 20,000.

Once a path has been specified, the Image Map or Event file can be created by invoking either the Export or Export As... commands. The Export command exports the ImageMap or Event file to the specified path. The Export As... command invokes an Export dialog in which the specified path may be redefined. An illustration of an exemplary Export dialog 22,001 invoked by the Export As... command is illustrated in Figure 22. The Export dialog 22,001 invoked by the Export As ... command may include a browse button 22,003 and a thumbnail illustration 4007 of the video with which the Image Map or Event file is associated. The browse button 22,003, known to persons skilled in the art, permits perusal of other predefined paths for exported Image Map or Event files.

In one embodiment, Map or Event files can be exported substantially simultaneously for all video files associated with a hypervideo 10,007 by invoking the Export All command. In one embodiment, Export dialogs 22,001 are displayed for the videos for which no Image Map path was previously defined.

In one embodiment, the commands for exportation, described above, may be invoked in the following manners:

Positioning the pointer over a thumbnail illustration 4007 of a specific media in the Media Warehouse window 20,003, and actuating a button, such as the Right mouse button to access the Export and Export As... commands;

Actuating a button, such as an Export button 20,011 or an Export All button 20,012, in the control bar 20,010 in the Media Warehouse window 20,003 to respectively invoke either the Export or Export All commands; and

Invoking the Export, Export As ..., and Export All commands in the media menu of the menu bar 20,005.

Transformation Process

Invocation of the Export commands causes the hotspot 8003 and notification frame 1157 data in the project file 1670 to be transformed respectively into the formats of the Image Map and Event files. If a project file 1670 (e.g. .obv file) is re-opened and edited after an initial transformation, or exportation, then transformation by invocation of the Export commands must be repeated.

One embodiment of the transformation process for Image Map files will now be summarized. Unlike hotspot 8003 defined in a project file 1670, a hotspot 8003 in a Image Map file may only have a fixed geometry. Therefore, a hotspot 8003, whose geometry varies among frames, in a project file 1670 must be redefined as two or more hotspots 8003, each of whose geometry is fixed, in the corresponding Image Map file. A hotspot 8003, defined in the Image Map file, may have a fixed geometry over two or more frames.

Hotspot(s) 8003 will not be translated by the invocation of an Export command if a hotspot Enable flag is set off in a General page 21,002 of a corresponding Hotspot Properties sheet 21,101. Thus, such hotspot(s) 8003 will not be included in the Image Map file created by the transformation process. The default length of those frame sequences will be determined in accordance to the transmission bandwidth, as described above. A new MAP range of frames will be used for the first and last frames in which a hotspot 8003 appears to ensure that the hotspot 8003 is only provide for frames in which the hotspot 8003 was originally defined. The number of MAP statements created by the modified authoring tool for RealVideo (TM) 20,001 is determined by the available bandwidth of the client-server network used to perform the hypervideo 10,007.

Creation of RealVideo (TM) Multimedia Files including Hypervideo Data

In one embodiment, each Image Map or Events file must then be encoded into an .rm file using the RealVideo (TM) rmmerge tool. The resulting .rm file must then be encoded, or exported, into a corresponding, already encoded, media (e.g. video) file again using the rmmerge tool to create a final .rm file. In another embodiment, the foregoing process is performed by the user who must twice run the rmmerge tool.

In a further embodiment, this process is automated. The modified version of the authoring tool for RealVideo (TM) 20,000 not only exports the hotspot 8003 and notification frame 1157 data, as described above, but also causes the rmmerge tool to be executed so as to create the final .rm file. To execute the rmmerge tool, the modified version of the authoring tool for RealVideo (TM) 20,000 utilizes dynamic link libraries (DLLs), rmimap.dll and rmevents.dll, and the RealVideo (TM) software development kit (SDK). The modified version of the authoring tool for RealVideo (TM) 20,000 utilizes the RealVideo (TM) SDK to encode the video file into a RealVideo (TM) .rm video file format.

When implementing the automated process, an export dialog for automatic translation 23,001, illustrated in Figure 23, may be used. The export dialog for automatic translation 23,001 is displayed upon executing an export command.

The export dialog for automatic translation 23,001 permits either temporary or permanent .rm files including Image Map and/or Event file data to be created. Permanent .rm files for Image Map and Event file data may be created upon checking the *Use intermediate files* box 23,101. Upon actuating the create file/s button 23,103, permanent .rm files may be created for Image Map and Event file data may be files if their respective *Build* boxes 23,105, 23,107 are checked. To create the permanent .rm files including Image Map and Event data, temporary Image Map and the Events files are first built as temporary text files. Then, the permanent .rm files are created for the Image Map and Events file data, using the rmmerge tool. Respective names 23,109, 23,111 for the permanent .rm files for Image Map and Events data are specified in the export dialog for automatic translation 23,001. An encoded video .rm file is also created from the original video file using the RealVideo (TM) SDK only if the build box is checked. Because a video file may take minutes to encode, the build box may be left unchecked to avoid unnecessarily repeating video file encoding every time hotspots 8003 are encoded.

The name 23,117 for the encoded video .rm file is specified in the export dialog for automatic translation 23,001. After the permanent .rm file(s) are built, the video .rm file is merged with the permanent .rm file(s), using the rmmerge

tool, if at least one corresponding *Merge* box is checked, to create the final .rm file. The name 23,119 for the final .rm file is specified in the export dialog for automatic translation 23,001. If neither *Merge* check boxes 23,113, 23,115 is checked, then the final .rm file is not created.

Alternatively, if the *Use intermediate files* box 23,101 is not checked, the video file is encoded into a temporary .rm file using the RealVideo (TM) SDK. If the Image Map file *Merge* box 23,113 is checked, then the Image Map data is translated into a temporary text file, which is translated into a corresponding temporary .rm file. If the Events *Merge* box 23,115 is checked, then the Events data is translated into a temporary text file, which is translated into a corresponding temporary .rm file. Then, the video .rm file is merged, using the rmmerge tool, with the specified temporary .rm files to create the specified final .rm file. If both of the Merge check boxes are unchecked, the video file is simply encoded into the final .rm file.

Encoding Properties

In one embodiment, the export dialog for automatic translation 23,001 may include an Encoding Properties Sheet 24,000, illustrated in Figure 24A. The Encoding Properties Sheet 24,000 permits entry of the data required by the RealVideo (TM) Encoder. The Encoding Properties Sheet 24,000 may be accessed by actuating the Encoding Properties... button 23,121.

The Encoding Properties Sheet 24,000 includes the Encoding Data page 24,001, illustrated in Figure 24A, and the Encoding parameters page 24,002, illustrated in Figure 24B. In another embodiment, the Encoding Data page 24,001 and the Encoding parameters page 24,002 may be added to the Media Properties Sheet.

The Encoding Data page 24,001 may contain general data that will be inserted into the video .rm file. The general data may include the title 24,003, authorship 24,005, and copyright information 24,007 associated with the hypervideo 10,007 created in the RealVideo (TM) format.

The Encoding Parameters page may contain parameters for the video encoding process, including:

a *Template* box 24,101 from which standard RealVideo templates can be selected. By selecting a standard template, other data fields on the Encoding Parameters page 24,002 may be selected according to the data associated with the selected template.

Changing one of the other fields will set the template field to the appropriate value (if there is no template with current parameters, *Custom* will be set).

Multiple Video Files

The modified version of the authoring tool for RealVideo (TM) 20,000 may be used to define and manipulate hotspots 8003 and notification frames 1157, including defining target(s) 7003, in multiple video media files substantially simultaneously. The hotspot 8003 and notification frame 1157 data may then be exported into final .rm files corresponding to each of these video files. using the process described above, in a manner similar to the Save/Save All/Save As... feature in a multi-document application.

2.4 Dynamic Hypervideo

Previously, hypervideos 10,007 have been suggested as having fixed or static parameters. Static hypervideo 10,007 parameters can not be changed during the performance of the hypervideo 10,007. One example of a static project file parameter is the parameter that defines a target 7003 media file linked to a media object. Thus, the target 7003 media file can not be changed during the performance of the hypervideo 10,007. As a result, a viewer of a performing hypervideo 10,007 is limited to navigating through media files defined during hypervideo 10,007 authoring.

To enhance the robustness and versatility of hypervideos 10,007, it is desirable to create dynamic hypervideos by allowing the hypervideo 10,007 parameters to vary during hypervideo 10,007 performance. Thus, for example, the hypervideo story board 1116 may change during hypervideo performance.

Dynamic hypervideos are hypervideos 10,007 having hypervideo 10,007 parameters that may vary during hypervideo 10,007 performance. The hypervideo parameters may include, but are not limited to, parameters describing media objects (e.g. hotspots 8003 and notification frames 1157), targets 7003, target levels 7601, cursors, and hypervideo 10,007 display window size. For example, by varying parameters, targets 7003 of media objects may be changed, media objects may be disabled so that the media objects may not be actuated, and hypervideo 10,007 display window size may be altered during hypervideo 10,007 performance.

The hypervideo 10,007 parameters may be varied during hypervideo 10,007 performance as a result of communications from application programs, and data retrieved from databases or log files. Viewer interactions may also indirectly vary hypervideo 10,007 parameters as described below. Viewer interactions with the hypervideo 10,007 may occur, for example, when a viewer actuates a hotspot 8003, or just positions a pointer over a hotspot 8003. Dynamic hypervideos that communicate with application program(s) may be known as hypervideo applications. Hypervideo applications may be used for example for electronic commerce and corporate training, and will be further described below.

In one embodiment, dynamic hypervideos may be created with a hypervideo system 25,000, depicted in Figure 25. The hypervideo system 25,000, illustrated in Figure 25, includes multiple servers, each having one or more components (e.g. database 25,037, project file 1670, application program 25,038, etc.) A client 25,003, having a player 25,039, may be coupled by a first network 25,050a to a first server 25,005a that may have a dynamic hypervideo server 25,035, logging tools program 25,047, log file 25,049 (e.g. a log database and/or a log file) and a project file 1670. The client 25,003 is also coupled by a second network 25,050b to a second server 25,005b that may have a media server 25,045. The media server 25,045 may include a video server and/or a web (or HTML) server. The first server 25,005a is also coupled by a third network 25,050c to a third server 25,005c that may have application program(s) (e.g. third party applications and/or third party servers) 25,038. The first server 25,005a is

also coupled by a fourth network 25,050n to a fourth server 25,005n that may have database(s) (e.g. database severs) 25,037. In one embodiment, the player includes a modified run-time module for receiving and processing streamed hypervideo data, and a media player (e.g. RealNetworks (TM) media player) for receiving and projecting streaming audio and video to a viewer. In one embodiment, the modified run-time module controls the media player.

In another embodiment, each network 25,050a-25,050n may use the Internet or an intranet, or a combination thereof. In yet another embodiment, the hypervideo system 25,000 may include multiple clients 25,003 coupled to the first and second servers 25,005a, 25,005b. The additional clients may each include a player 25,039. In one embodiment, an additional client 25,003 may be coupled to the first server 25,005a, and includes a modified authoring tool for creating a dynamic hypervideo. In one embodiment, the modified authoring tool may be a modified version of the authoring tool 1001, described above, or a variant thereof. The modified authoring tool may be used to create the project file 1670 that can be transferred to the first server 25,005a electronically across a network or physically with a storage device, such as a compact disk-read only memory (CD-ROM).

In further embodiments, the hypervideo system 25,000 may have fewer or more servers upon which the components are respectively consolidated or further distributed. In another embodiment, the servers and client(s) may be interconnected in a different manner, for example directly coupled to one another. In yet a further embodiment, the database(s) 25,037 may include multiple databases, each of which operates with one or more components of the hypervideo system 25,000.

One embodiment of the operation of the hypervideo system 25,000 will now be described. In this embodiment, the video and hypervideo data streams may be wholly separated. For example, hypervideo data may be streamed from the dynamic hypervideo server 25,035 to the client 25,003. Video data may be separately streamed from the media server 25,045 to the player 25,039.

In another embodiment, some hypervideo data may be exported into a multimedia (e.g. video) file by a modified authoring tool. The media and

hypervideo data in the multimedia file may be streamed from the media server 25,045 to the player 25,039. For example, the media server 25,045 may stream hypervideo data, including media object identifier data, time or frame data, shape data, and geometry data (e.g. layer one of multilayer hypervideo data stream described above), to the player 25,039. This technique may increase second network 25,005b transmission capacity consumption, but may reduce first network 25,005a transmission capacity consumption and dynamic hypervideo server 25,035 processing requirements.

In one embodiment, some hypervideo data may be processed in the dynamic hypervideo server 25,035, and not streamed to the client 25,003, to reduce the player's processing and memory requirements, and to efficiently facilitate dynamic hypervideos. The interaction between the player 25,039 and the dynamic hypervideo server 25,035 will now be further illustrated.

In one embodiment, the client 25,003 communicates with the dynamic hypervideo server 25,035 by firing events, further described below, to a dynamic hypervideo server 25,035. Upon receiving the fired event, the dynamic hypervideo server 25,035 may respond by sending a command, which may include hypervideo data, to the client 25,003. For example, if a viewer actuates a hotspot 8003 during the performance of a media in the player 25,039, the client 25,003 may fire an event to the server indicating that the hotspot 8003 has been actuated. In response, the dynamic hypervideo server 25,035 may stream a command, including target 7003 data identifying video files to be executed, to the player 25,039. The player 25,039 may then stream a request to the media server 25,045 that the identified video files be streamed to the player 25,039 so that the video files may be performed. Other techniques for varying project file parameters will be later described.

Dynamic Functions Provided by the Hypervideo System

Dynamic functionality provided by a hypervideo system 25,000 will now be further described. Hypervideo systems 25,000 may provide functions found in hypervideos 10,007, described above. Thus, like hypervideos 10,007, dynamic hypervideos may include hotspots 8003, event notification 1114, and markers

1155. Additionally, however, these features may be modified during the performance of the dynamic hypervideo, for example, based upon information derived from the database 25,037, application program 25,038, a log file 25,049, or user interaction.

Dynamic functions provided by the hypervideo system 25,000 may include:

a. Hotspot Feature Availability. Hotspot 8003 features may be enabled or disabled according to data retrieved from a database 25,037, an application program 25,0389, a log file 25,049, or due to viewer interaction. Different hotspots 8003 features may be affected, for example, visual marking of hotspots 8003 or execution of target(s) 7003 may be disabled.

b. Targets. Targets 7003 linked to hotspots 8003 may be modified by removing or adding targets 7003, or by changing the execution and display parameters of target(s) 7003. For example, a dynamic hypervideo may be an interactive advertisement for a department store which may be viewed by customers. The dynamic hypervideo server 25,035 may ascertain the identity of customers viewing the dynamic hypervideo. The dynamic hypervideo server 25,035 may also query a database 25,037 to determine whether the customer holds the department store's credit card. The customers may point and click on an illustration of department store's credit card. Based upon the query results, the dynamic hypervideo server 25,037 may cause monthly special offers to be displayed to customers that are credit card holders. Also based upon the query results, the dynamic hypervideo server 25,037 may cause information illustrating the privileges of a holder of the department store credit card, and about how the credit card may be acquired, to customers who do not hold the department store's credit card.

c. Project Settings. General attributes of the dynamic hypervideo, including initial settings, such as base target(s) 11,001 and display parameters, may be modified.

d. Media Properties. Media elements 1690 of the dynamic hypervideo may have general properties unrelated to targets. Such properties may include:

1) Audio track. Videos may include corresponding audio spoken in different languages and thus stored separately, for example, on different tracks. For example a dynamic hypervideo performing in a French speaking environment may adapt to broadcast French language audio; and

2) Restricted access videos and video versions. Based upon externally supplied information, a dynamic hypervideos may ascertain which, if any, version of a media, such as a video, a viewer is authorized to observe. For example, a dynamic hypervideo configured for children may only display appropriately rated versions of media.

e. Markers. Markers 1155, which may be used to highlight media objects, or may convey additional information associated with the media object, may be modified. Such markers may be overlay images, text, or special filters. For example, a training hypervideo application may include a hypervideo simulation of an industrial plant. The dynamic hypervideo may display relevant information, such as temperature and humidity, as markers 1155 next to equipment in the industrial plant. The markers 1155 may display the temperature and humidity data in real-time. This is desirable because, for example, in the industrial plant simulation, the temperature may constantly change in response to trainee interactions with the dynamic hypervideo. The marker 1155 displaying temperature data may obtain the temperature data from a database 25,037.

Dynamic hypervideos may also support notification frames 1157. Notification frames 1157, or time based events, may be implemented in media, including hypervideo applications. Notification frames 1157, as described above, permits information, including text, graphics and video, to be displayed in synchronization with a timebase (e.g. by frame unit or time unit) of a video to enrich the information content of a dynamic hypervideo. The hypervideo system 25,000 may execute a target 7003 when a hypervideo 10,007 reaches a certain time or frame (e.g. notification frame 1157). A target 7003 may include any of those previously or subsequently described, including an image, a video, or an executable program. Thus, for example, while a hypervideo application is displayed, slides containing relevant information may be displayed concurrently with notification frames 1157. In one embodiment, notification frame 1157 target or frame parameters may be altered during the performance of a hypervideo.

Dynamic Data Sources

Data for dynamic project file parameters may be provided from the following sources:

Databases. Commonly available databases may be implemented in the hypervideo system 25,000. However, the hypervideo system 25,000 may be modified to accommodate other databases. Object Database Connectivity (ODBC), a standard windows applications program interface (API), may be used to facilitate communications, for example, between the dynamic hypervideo server 25,035 and a database 25,037. The hypervideo database may obtain data from a database 25,037 by issuing a query to the database 25,037. During query authoring, placeholders (e.g. variables) whose contents are determined during hypervideo application execution, may be used in the queries. In one embodiment, the queries may be standard query language (SQL) queries.

Applications. Hypervideo applications may be implemented with the

hypervideo system 25,000 including an applications program 25,038. In one embodiment, the application program(s) 25,038 may receive and/or generate data in real-time. Other hypervideo system 25,000 components, including the dynamic hypervideo server 25,035, may serve as sources and/or destinations for such data associated with the application programs(s) 25,038. In one embodiment, application programs(s) 25,038 may directly communicate with the hypervideo system 25,000 components through SDK-like interfaces. Such communications may be facilitated with ActiveX commands and events, and scripting languages such as VBScript or JavaScript. Communications between application program(s) 25,038 and other hypervideo system 25,000 components may be stored and retrieved from an intermediary file. In one embodiment, data may be transferred between application programs(s) 25,038 through a database 25,037, described above.

In one embodiment, an application program 25,038 may send commands or data to a dynamic hypervideo server (and thus to a hypervideo 10,007) through call back objects or APIs. The application program 25,038 commands or data may be used to manipulate project file parameters during dynamic hypervideo performance. The dynamic hypervideo may communicate with the application program 25,038 by firing events, for example through a call back object, to the application program 25,038. In another embodiment, a player 25,039 and an applications program 25,038 may communicate with one another directly, if a network connection exists, or indirectly through the dynamic hypervideo server 25,038.

A hypervideo application may also include one or more application programs 25,038 that generate graphical output prepared for display. The hypervideo application may access the application program(s) 25,038 to obtain the graphical output, and may do so, in a manner similar to the one described above, using placeholders.

If the graphical output is available, it may then be displayed. An example of this technique will now be described. A kiosk may include a hypervideo system 25,000 for performing a dynamic hypervideo application that provides worldwide weather forecasting information. The dynamic hypervideo application may display a turning globe. A viewer may point and click on a region of the globe. Then, the dynamic hypervideo application may query a weather forecasting database to obtain next week's forecasted weather data for the selected region. The retrieved forecasted weather data may then be formulated in a bar chart by a charting application. The dynamic hypervideo application then may automatically display the bar chart over the selected region of the turning globe.

Log File The hypervideo system 25,000 may permit hypervideo 10,007 activities to be logged, by the dynamic hypervideo server 25,035, during hypervideo 10,007 performance. In one embodiment, an author of a hypervideo application may specify, during hypervideo application authoring, which viewer interactions (or other hypervideo application activities) and corresponding information are to be logged. During hypervideo application performance, the logged information may be retrieved by the dynamic hypervideo server 25,035 to manipulate the performance of the hypervideo application.

Hypervideo Applications

Private, governmental and commercial organizations are becoming reliant upon network application programs that facilitate interaction within or outside the organizations. In order to make the application programs more compelling and intuitive to users, the application programs should include dynamic content. For example, application programs may be enriched by including visual media, including graphical and animation. Application programs may be made more intuitive to users by allowing the users to interact with the visual media, such as with videos.

With hypervideo applications viewers can navigate and access a variety of databases, forms, etc., including knowledge databases, purchasing forms, instructional material or any other services provided by the application. Exemplary hypervideo applications include:

Electronic Commerce Application

Businesses, ranging from small retailers to large manufacturers, are availing themselves of the Internet and other networks to promote, market and sell their services and products. Electronic commerce solution providers offer toolkits for developing and managing on-line web sites, stores and catalogs. Such toolkits may facilitate storefront creation, transaction processing, and electronic distribution of software or data. Lack of potential customer interest in sites created with such toolkits has lead businesses to look for more compelling web sites. Video can be used to make such web sites more alluring to enhance potential customer interest and enhance product promotion.

However, it may be desirable to closely integrate the video with the purchasing process. For example, it may be preferable for a potential customer to point and click on an item (e.g. media object) in a hypervideo 10,007 to initiate the purchase of the displayed item.

The purchase process may be implemented with a hypervideo application provided by the hypervideo system 25,000 including an electronic commerce application program. In one embodiment, the hypervideo application may be dynamic. A web site may include the dynamic hypervideo application whose parameters may vary, for example depending upon date or time, or depending upon the customer accessing the web site. Further, the hypervideo system 25,00 may record the activities performed by customers during their visits to the web site.

An exemplary electronic commerce hypervideo application will now be described. Potential customers may view and interact with a hypervideo application on a web site of a clothing store. The hypervideo application displays a fashion show in which models exhibit garments while walking down a runway. A potential customer may point and click on a garment worn by a model to obtain

information associated with the garment, and to obtain an order form which can be completed to buy the garment.

The hypervideo application may be dynamic. The information, displayed on a player 25,039 when a potential customer points and clicks on a hotspot associated with a garment, may vary in real-time. For example, the price of the garment may be varied in real-time. When the potential customer points and clicks on the garment hotspot, the dynamic hypervideo server 25,035 may query a database 25,037 including the garment prices. The garment prices in the database 25,037 may be varied in real time. If a potential customer positions a pointer over the garment hotspot at least a certain number of times (causing event messages to be fired from the player 25,039 to the dynamic hypervideo server 25,035 each time the pointer is positioned over the garment hotspot), the dynamic hypervideo server 25,035 (upon receiving at least the certain number of event messages) may cause a discounted sales price to be displayed for the garment. The dynamic hypervideo server 25,035 may then facilitate the customer's purchase of the garment at the discounted sales price .

Training Application

Many organizations utilize Intranets to provide training services to their employees, or other parties. Parties may participate in prepared lessons, practice drills, or view live demonstrations. Video is already being used in training, for example to display live demonstrations.

Hypervideo applications, however, enhance the allure and accuracy, and hence the effectiveness, of training. An exemplary hypervideo application for training will now be described. A trainee may view a hypervideo application that displays an experiment conducted in a laboratory of a chemical plant. The trainee may point and click on one of the chemicals reacting during the experiment. As a result of the trainee's action, data obtained by the dynamic hypervideo server 25,035 from a database 25,037 or an application program 25,038 at that point in time during the experiment may be displayed.

Media-Intensive Application

Organizations, for example entertainment businesses, news providers, and government agencies, may store large quantities of video which may be retrieved over networks, such as intranets, the Internet, or combinations thereof. In one embodiment, the videos may be stored in a video database 25,051. For example, news services may store numerous videos of past and present news stories on servers. A hypervideo application may be formed from the videos to permit viewers to interact with the 'videos'. For example, a viewer may point and click on a person or item displayed in a video so that further information, such as a snapshot or related articles, about the selected person or item is displayed to the viewer.

Interactive Advertising Application

Interactive advertising, for example online advertising, has unique capabilities in comparison to other forms of advertising. Viewers of interactive advertising may readily access further information about an advertised product or service, or conversely readily ignore such advertising. Further, interactive advertising applications may record data relevant to advertising effectiveness, including the type and frequency of information sought by viewers. For example, the recorded data can be used to bill advertisers based upon the information displayed to the viewers.

Interactive advertising may be implemented in a more effective and alluring manner with dynamic hypervideo applications. An online advertisement hypervideo application may permit interactive advertisements to be altered in response to viewer actions.

An exemplary dynamic hypervideo advertising application will now be described. While watching an advertisement, such as an automobile advertisement, a viewer may point and click on a hotspot associated with a car, shown driving along a picturesque shoreline highway, to obtain further information about the car. Other targets, executed when the car hotspot is actuated, may include a purchase order form or a chart of current revenue figures related to sales of the car model.

Multi-Purpose Application

Hypervideo applications may have multi-purposes, for example electronic commerce, advertising, and training. An illustration of a multi-purpose dynamic hypervideo application follows.

A kitchen appliance retailer implements an online store to sell kitchen appliances. Their web site permits viewers to look at and purchase items used in a kitchen, including cutlery and cabinet sets. The web site includes:

1. Interactive sections, or attractions, including a recipe corner in which renowned chefs demonstrate cooking their favorite dishes, and the utensil corner in which cooking utensils are demonstrated.

2. Knowledge database of kitchen appliances and other kitchen related issues.

3. Purchasing facilities permitting on-line purchasing of culinary goods. Payment and billing may be performed with a credit card on-line, or through the mail. Purchased goods may be mailed to the customers.

4. A home page, for regular customers of the kitchen appliance retailer, where the regular customers may obtain special purchase offers, or receive guidance about designing their kitchens.

5. Kitchen–zine. An interactive e-zine about cooking and kitchens, linked directly to the web site's purchasing facilities.

Another service for regular customers may also include creation and maintenance of online wedding gift lists.

The kitchen appliance retailer's computer system may include aa database 25,037, a system administrator, a media server 25,045, an internal local area network (LAN) 25,050, and a dynamic hypervideo server 25,035.

The interactive recipe corner may include streaming hypervideos of chefs preparing their favorites dishes. The hypervideos 10,007 may be embedded inside HTML pages, which may include more information about the corresponding recipe and the final product. The HTML page may change as dish preparation proceeds. Pointing and clicking on a product found in the hypervideo 10,007 may cause more information to appear (e.g. pop-up) about the product. Pointing and clicking on an appliance may cause a query of the retailer's database to obtain

information about the appliance. The information may be displayed to the user as an HTML page in a frame separate from the video. The appliances, found in the retailer's current inventory, may be marked (e.g. highlighted) in the hypervideo application. Appliances which are on special sale may be marked with SALE tags. Pointing and clicking on a hotspot corresponding to an appliance may display, instead of the originally viewed media, a demonstration video of the appliance. The actuation of the appliance hotspot may also cause another target 7003 to be executed. For example, a URL flip may be executed to display details about the selected appliance in another frame. The newly displayed video may also include a small image on it reciting BUY NOW. Upon pointing and clicking upon the BUY NOW image, an order form for the corresponding appliance may be displayed to permit the viewer to purchase the appliance from the retailer.

Other Hypervideo Application Uses

Hypervideos 10,007 and hypervideo applications may be used to provide interactivity for the following: corporate training, distance/web-based learning, multimedia display kiosks and point of sale terminals, video archive retrieval, electronic commerce (e.g. online stores), security systems, medical analysis and treatment systems, entertainment systems, advertising, hotel information provision, real estate sales, architectural design demonstration, industrial tele-operation, home video display, adult entertainment display, customer support automation, catalogs, sales force automation, and corporate communications.

Database Integration and External Data Referencing

In one embodiment, data for dynamic hypervideo parameters may be extracted from a database 25,037. Data extracted from database(s) 25,037 and application(s) 25,038 may be in a raw format, such as binary numbers or text, or in a processed format, such as graphics or a video. In one embodiment, data from external sources (e.g. database(s) 25,037, application program(s) 25,038, log file(s) 25,049), may be integrated into a hypervideo application in a two step process. First, data from an external source may be accessed, and relevant data retrieved by the hypervideo system 25,000. Second, the retrieved data can either

be processed for display, or provided, for example to the dynamic hypervideo server 25,035, to control dynamic hypervideo performance.

In another embodiment, a hypervideo 10,007 may be used as an interface to permit a viewer to access data from a database in an intuitive manner. For example, in a hypervideo application of a clothing store, dresses, for which hotspots 8003 are defined, may be marked with tags displaying their price. Such prices may updated in real time through the clothing store's database. In another example, a web site may provide support for truck engine mechanics. Videos of truck engines may be displayed through the web site. The videos include hotspots 8003, which allow viewers, such as truck engine mechanics, to point and click on truck engine parts to obtain information about those parts. The site maintains a database of registered truck engine mechanics. Only truck engine mechanics that match the web site's registrations may be allowed to obtain information about the parts, linked to hotspots associated with the parts, and displayed when the registered truck engine mechanics point and click on corresponding hotspots 8003. Others, not identified in the database, may only view the video, and may not access the part information.

Database Access During Authoring

In one embodiment, an authoring tool 1001 may access the database 25,037, including table structures and data, when a hypervideo application is authored. In another embodiment, the authoring tool 1001 may partially access the database 25,037 so that table structures, but not data, are available to the authoring tool 1001 when a hypervideo application is authored.

Integration of A Dynamic Hypervideo with Multimedia and Video Databases

Databases including multimedia and video are special types of external databases, and may also be used in a hypervideo 10,007. Databases, including multimedia and video databases, may be accessed to dynamically insert new media in a hypervideo 10,007 during the performance of a hypervideo 10,007. Videos or pictures, that were not originally authored in a hypervideo 10,007, may be displayed during the performance of the hypervideo 10,007.

For example, a broadcasting network having a large database of reportage video, in which images of persons are marked. A viewer may point and click on a person's image to access the broadcasting network's database to retrieve the most recently available photograph of the person. The photograph may then be displayed side by side with the currently playing hypervideo 10,007. Multimedia and video database data types, used in hypervideo 10,007s, may include video, images, sound, music, cursors, icons, etc.

Database Storage of Hypervideo Data

In one embodiment, hypervideo 10,007 data may be stored in a database, such as a conventional database, rather than in a project file 1670 which may use a proprietary format. As a result, the hypervideo 10,007 data, such as the hypervideo 10,007 data corresponding to media and media objects, may be readily accessed, for example by a standard query language (SQL) query. The hypervideo 10,007 data in a database may be accessed and manipulated to modify hypervideo 10,007 performance and/or to permit logging or the acquisition statistics pertaining to hypervideo 10,007 performance. The hypervideo 10,007 statistics may be derived from user interactions with a hypervideo 10,007. The hypervideo 10,007 data stored in a database 25,037 may include the range of frames in which a hotspot 8003 is defined, location of notification frames 1157 in a video, the number of video frames displayed, or the number of times a hotspot 8003 has been actuated.

Data Logging and Analysis

In another embodiment, the hypervideo system 25,000 may also log information about and during the performance of a hypervideo 10,007. To this end, the hypervideo system 25,000 includes logging tools 25,047 that permit:

1. Logging requirements for a hypervideo 10,007 to be defined.
2. Logging information to be analyzed during or after the performance of the hypervideo 10,007. When analyzed during the performance of the hypervideo 10,007, the logging information may be used to alter the parameters of the hypervideo 10,007.

3. Analyzing offline statistics about logging information gathered during hypervideo 10,007 performance (e.g after or during hypervideo 10,007 performance).

In one embodiment, when a hypervideo 10,007 is authored, the logging requirements, or data to be logged, may be specified. The data to be logged may be specified by the authoring tool 1001, or with logging tools 25,047. The logged data may include the hypervideo 10,007 activities that initiate logging, and the corresponding information to be logged. Data to be logged may be customized for each hypervideo 10,007. For example, a hypervideo 10,007 may create data that logs each hotspot 8003 that is activated when a viewer points and clicks on the hotspot 8003. The hypervideo 10,007 may also log each occurrence of a pointer being positioned over a specified hotspot.

A hypervideo scripting language, further described below, may be used to define the activities and the corresponding information to be logged during the performance of a hypervideo 10,007.

The logging tools 25,047 may permit application managers to statistically analyze and interact with a hypervideo application based upon the logged data. So that it may be readily accessed by application programs (s) 25,038, and their managers, logged data may be stored in logging file(s) 25,049 in a standard format. For example, logged data may be stored in logging file(s) 25,049 in database formats supported by the application program(s) 25,038.

The hypervideo system 25,000 may log data pertaining to a specific viewer's interaction with a hypervideo 10,007. Thus, an application administrator may be informed of the viewer's interaction with the hypervideo 10,007. The hypervideo 10,007 may also log data pertaining to specific interactive features (e.g. hotspots 8003) actuated by any viewer of the hypervideo 10,007.

Authoring Tool GUI for Hypervideo System

In one embodiment, the authoring tool GUI 26,001 for the hypervideo system 25,000, illustrated in Figure 26, is similar to the authoring tool GUIs 10,001, 15,001, 20,001 described above, but may have additional features. One

additional feature may be that the authoring tool permits more windows to be displayed simultaneously. Some of the other extra features in the authoring tool have their own window. For example, in addition to the Media Warehouse window 26,003 and the Workshop window 26,011, the authoring tool GUI 26,001 for the hypervideo system 25,000 may include a Queries window and a Handlers Notebook window, later described.

Any window, such as window 'x', may be displayed by actuating a "View 'x' Window" button on the control bar 26,007 of the authoring tool GUI 26,001 for the hypervideo system 25,000. The authoring tool GUI 26,001 for the hypervideo system 25,000 will now be described in further detail.

Media Warehouse Window

The Media Warehouse window 26,003 may support media file types including: Video, Image, Text, Sound, Music, HTML, Executable, Animation, Virtual Reality Modeling Language (VRML), QuickTime VR (TM), and SQL queries used as targets.

Database Integration

In one embodiment, a database 25,037 that is object oriented may be used to enhance the user-friendliness of the authoring tool GUI 26,001 of the hypervideo system 25,000. Queries to the database 25,037, or other external data sources including applications 25,038, may be objects, which may be created, edited, manipulated, and referred during the authoring process. These query objects may be used as targets, linked to media objects, that for example may require a hypervideo's viewer to provide information so that the query object may retrieve data from a database 25,037 or other external data source. The retrieved data may be used to vary parameters of a hypervideo 10,007.

Queries Warehouse Window

The authoring tool GUI 26,001 of the hypervideo system 25,000 includes a Queries Warehouse 26,072 and corresponding Queries Warehouse window 26,071. Query object elements may reside in the Query Warehouse 26,072.

Icons 26,073 of Query object elements may reside in the Queries Warehouse window 26,071. The Queries Warehouse may function analogously to the Media Warehouse 1003. Thus, the query object parameters defined in a dynamic hypervideo may be saved with other dynamic hypervideo parameters, for example, in a project file or in a database 25,037.

The Queries Warehouse window 26,071 may function analogously to the Media Warehouse window 26,003. The query object elements may be associated with query objects. The Query Warehouse window 26,071 may permit the query objects to be manipulated.

The Query Warehouse window 26,071 may include a control bar 26,075 that includes relevant buttons, for example buttons for Properties 26,077, New Query 26,079, Detailed List 26,081, etc. The Queries Warehouse window 26,071 may be displayed or hidden as desired by the authoring tool GUI 26,001 user.

Creating a New Query

In order to create a query object, or to import a query object from another file, the New Query button 26,079 may be actuated, such as by pointing and clicking. Actuation of the New Query button 26,079 may cause a Query dialog to be displayed. The query corresponding to the query object may then be defined, for example, in a step by step manner, in the Query dialog, or wizard. Upon defining the query in the Query dialog, a corresponding query element may be placed in the Query Warehouse 26,072, a corresponding icon 26,073 of the query element may be placed in the Query Warehouse window 26,071, and a corresponding query object is created. The query associated with the query object may be later edited using the Query Properties page which is later described.

In one embodiment, when accessed, successive pages of the Query dialog are displayed to permit the step by step entry of the query. In another embodiment, the query may be defined with the standard query language (SQL). In another embodiment, only common or basic SQL commands may be supported

by the Query dialog. In yet another embodiment, more complex SQL queries may be implemented by allowing the SQL queries to be textually edited.

The first displayed page is the Data Source page 27,001 of the Query dialog 27,000, illustrated in Figure 27. The source of data 27,003 upon which a query operates may be defined in the Data Source page 27,001. In one embodiment, any SQL database, having an ODBC driver installed, may be selected. Upon specifying a source of data 27,003, the database file upon which the query operates may be defined in the Data Source page 27,001.

Then, the query may be defined. The query may be a predefined query selected from predefined queries listed on the Query page 27,005. If selected, the parameters of the predefined query may be provided to the query object, and may be later altered.

If a predefined query is not chosen, then the query may be defined. To define the query, columns 28,005 to be used in the query may be first selected from available columns 28,005 in the selected database file using the Choose Columns page 28,000, illustrated in Figure 28. The Choose Columns page may also be used to display tables in the selected database file.

Upon selecting columns 28,005, the query may be defined in the Filter Data page 29,000, illustrated in Figure 29. In the Filter Data page 29,000, the query may be defined by combining some or all selected columns 28,005 with logical operators 29,003.

The query 30,001 may be reviewed and edited in another page, the SQL page 30,000, illustrated in Figure 30. Further, in the SQL page, the Access Database option 30,003 may define the frequency of database file access, permitted by the query 30,001, to extract data from the database file. In one embodiment, the database file may be accessed everytime the query 30,001 is used so that current database information is provided in response to the query 30,001. In another embodiment, the database file may be accessed a limited number of times, such as just once, for example when the query 30,001 is first used, or prior to dynamic hypervideo performance. Upon actuating the "Advance" button, a more sophisticated query editor may be displayed. In the more sophisticated query editor, more complex queries may be created, reviewed

and edited. Upon defining a query 30,001, the resulting query 30,001 may be used to extract data, that satisfies the query 30,001, from the database file.

Queries Property Page

The Queries Properties sheet may be accessed, or opened, by actuating, such as by pointing and double-clicking on the corresponding query element icon 26,073, or by selecting the query element icon 26,073 and actuating (e.g. clicking) the Properties button 26,077. The Query Properties sheet may include general information about a query 30,001, for example, the corresponding SQL statement, and default parameters which may be applied when using the query 30,001 as a target 7003 or to modify dynamic hypervideo parameters.

In one embodiment, the Queries Property sheet may include General, SQL, and Display pages. In the General page, the database 25,037 and database file, query name, and frequency of database file access may be shown. In the SQL page, the SQL statement may be displayed, an SQL statement editor may be provided, and the SQL dialog may be accessed. The Display page may include default display parameters that may determine how the results for the corresponding query object are displayed to a viewer. In another embodiment, the Queries Property sheet may be the same as the Queries dialog.

Query Objects

Query objects may be used in different ways for different proposes. In one embodiment, a query object may be a target 7003 linked to a media object, such as a hotspot 8003 or a notification frame 1157. The corresponding query 30,001 is performed when the media object is actuated.

In the Media Workshop window 26,011, the query object may be linked to a media object, including a notification frame 1157, by dragging a link icon 31,001 from a Targets page 31,000, illustrated in Figure 31, of a media object to the Query Warehouse window 26,071 and dropping the link icon on the desired query element icon 26,073. As may occur when linking other media to a media object or notification frame 1157, a Query Target (or Link) Properties page may be displayed. Parameters related to linking the corresponding query element, and

media object or notification frame may be defined in the QueryTarget Properties page. Parameters relating to how data retrieved by the corresponding query 30,001 are displayed may also be defined in the Query Target Properties page.

In another embodiment, the query object may also be used as a marker 1155 for a media object, for example to display the information retrieved by the query as a tool-tip for the media object. Thus, when the query 30,001 is performed, the information displayed by the marker 1155 may be updated.

In yet another embodiment, the query object may be used to modify dynamic hypervideo parameters. The query object may be a variable in a logical expression (e.g. Boolean expression). The data retrieved by the corresponding query 30,001 may be compared against a constant value or other query objects's data. The results of the logical expression may be used to modify dynamic hypervideo parameters. The logical expression may be used by a hypervideo object's event handler, described below.

Hypervideo Application Features

A dynamic hypervideos may be implemented in an object oriented manner. Thus, the dynamic hypervideo parameters, stored in a database 25,037 or project file 1670, may be implemented in an object-oriented manner.

A dynamic hypervideo may include the following object types:

1. Media element objects corresponding to media files, and having the media files's general properties.

2. Media objects corresponding to hotspots 8003 and notification frames 1157, as described above.

3. Target (Link) and Target (Link)-level objects, which may be used to define respectively targets 7003 such as media objects, and target levels. The target and target-level objects may define the actuation properties of targets 7003.

4. Query objects, as described above.

Marking a Media Object

In addition to the marking 1155 described above, a media object in a dynamic hypervideo may be marked with SQL markers, tool-tip markers, and text marking. As described above, the SQL markers may display the results of a query, for example in a tool-tip format, and may be proximate to the media object. Text marking may display text, for example with no background coloring, and may be proximate to the media object. Tool-tip marking may display text with background coloring, and may be proximate to the media object.

Target and Target Level Objects

Target objects may also be implemented with SQL queries, or SQL links, and with a scripting language, or automatic linking. Target level objects may defined all targets 7003 on a level 7601.

Hotframe Objects

A hotframe object is a type of media object corresponding to a notification frame 1157. Each frame in a video may either be or not be a notification frame 1157. Thus, each frame may contain one or zero hotframe objects. A hotframe object may be linked to targets 8003 in the same manner as a hotspot 8003. The hotframe object may be linked to multiple targets 8003 and may have multiple link levels. When displaying a frame in which a hotframe object was defined, the hotframe targets 8003 may be executed. To create a hotframe object, the corresponding frame is selected, and then the Create Hotframe button, in the Hotframes Toolbar, is actuated. The Hotframe Properties sheet is then displayed so that the hotframe object properties may be modified and the targets 7003 for the hotframe object defined. Once the hotframe object is defined, a hotframe icon 26,501 is displayed in the right-top corner of the corresponding frame. The Hotframe Properties sheet for the hotframe object linked to the frame may be accessed by actuating, such as double-clicking with a left mouse button, the hotframe object, or by clicking on the hotframe with the right mouse button and selecting properties from a displayed list.

A Hotframes page may be included in the Media Properties sheet of a media element. This Hotframes page may identify all the hotframes defined in that media, and display a thumbnail of the corresponding frame in which the hotframes are defined.

Hotframes Toolbar

In one embodiment, the authoring tool GUI 26,001 of the hypervideo system 25,000 includes a Hotframes toolbar. The Hotframes toolbar may be similar to the hotspot toolbar and provides similar functions to manipulate hotframe objects. The Hotframes toolbar may be a floating toolbar which can be docked to the borders of the Workshop window 26,011.

One embodiment of a Hotframes toolbar 32,000 is illustrated in Figure 32. The Hotframes toolbar 32,000 may include the following buttons:

Create Hotframe 32,001 - Actuating (e.g. by pointing and clicking) the Create Hotframe button may create a new hotframe object in the current frame. If there is already a hotframe object defined in the current frame, the Create Hotframe button 32,001 may be disabled.

Goto Hotframe 32,003 - Actuating (e.g. by pointing and clicking) the Goto Hotframe 32,003 button will cause the corresponding frame in which the hotframe, selected in the Hotframe Combo box, to be displayed in the Workshop window 26,011.

Hotframes Combo box 32,005 - The Hotframes Combo box 32,005 contains the titles of all hotframe objects defined in the media present in the Workshop window 26,011.

Hotframe Properties 32,007- The Hotframe Properties button may cause a Hotframe Properties page, for the hotframe object associated with the frame currently displayed in the Workshop window 26,011, to appear.

Remove Hotframe 32,009 -Actuating (e.g. by pointing and clicking) Remove Hotframe button 32,009 may delete the hotframe object, and all associated information, that was defined for the frame currently displayed in the Workshop window 26,011.

Hotframe Properties Page.

The Hotframe Properties sheet 33,000, illustrated in Figure 33, may include at least two pages, a General page 33,003 and a Target, or Links, page 33,005. The Target page 33,005 may be identical to the Target page in the Hotspot Properties sheet. The General page 33,003 includes general information about the corresponding hotframe object, including hotframe object name and description (which can be modified by the user), and the name of the media with which the hotframe object is associated. The General page 33,003 may also include an "Enable" check box, which if unchecked may disable the corresponding hotframe object (so that the corresponding hotframe may not be activated) without requiring that the hotframe object be deleted.

The frame with which the hotframe object is associated can also be changed in the General page 33,003. The slider 33,701 or the Next/Previous buttons 33,801 may be manipulated to select a new frame with which to associate the hotframe object. Then, the "Change frame" check box may be selected, or checked, to associate the hotframe object with the new frame. If the selected frame already has a hotframe object associated with the selected frame, then a dialog box may be displayed and may include the following options:

Cancel - Make no changes to selected frame.

Overwrite - Associate the hotframe object to the selected frame and delete the previous hotframe object which was previously associated with the selected frame.

Merge - Merge the two hotframes objects into a single, new hotframe object that includes the properties, such as the targets 7003, of the two hotframe objects. The new hotframe object may be associated with the selected frame.

Content Design for Large Scale Media Databases

Problems may arise when randomly accessing large video files during performance of a hypervideo 10,007. These problems may be overcome with off-line auto-tracking, off-line automatic scene-change recognition, and off-line automatic linking according to subjects.

Stage Management

To create a hypervideo 10,007, tools other than an authoring tool 1001 may be needed. In one embodiment, the authoring tool 1001 may only be used to create hypervideos 10,007. However, in one embodiment, additional features may be provided to the authoring tool 1001 so that it can be used to create additional functionality for hypervideo applications. Some of the additional features may include the ability to define what will happen around (e.g. in the background) of the hypervideo 10,007, provide menu options from hosting applications, and the ability to integrate HTML files.

Broadcasting and Recording

A performing dynamic hypervideo may be recorded by actuating a switch. The recorded hypervideo performance may then be later displayed offline in a linear fashion.

Dynamic Hypervideo Events, Commands, and Handlers

The hypervideo system 25,000 may support event driven programming. Events are actions, including viewer actions, which may be related to a dynamic hypervideo. In one embodiment, event driven programming may be implemented in an object-oriented manner. The hypervideo system's support for event driven programming may include an extensive bank of events and commands, and the ability to define event handlers, to receive event messages, with the authoring tool GUI 26,001 of the hypervideo system 25,000.

As with the run-time module 1001 (or any OLE object), communication between the run-time module 1001 and the surrounding environment is achieved by using events and commands. An object associated with the dynamic hypervideo, during the performance of a dynamic hypervideo, may produce and fire an event message that indicates that a certain event occurred with respect to the object. Hypervideo applications may communicate with application program(s) 25,038 by firing event messages to the application program(s) 25,038. A recipient application program 25,038 may be affected upon receiving and processing an event message. Further, event messages fired by a dynamic

hypervideo make the components of the hypervideo system 25,000 that receive the event messages aware of the dynamic hypervideo's status.

Commands may be sent by an application 25,038, or by an event handler in a hypervideo 10,007 to manipulate the performance of the same or another hypervideo 10,007. For example, the commands may alter hypervideo 10,007 interactive features or other parameters, or may initiate the performance of a media within the hypervideo 10,007. Commands may be sent to a hypervideo 10,007 as a result of the user interaction with another hypervideo 10,007, or in response to an event triggering the command from an event handler.

Objects in a hypervideo 10,007 may include media objects, including objects for hotspots, and objects for media, targets, target-levels, and frames. Each class, that instantiates an object in a hypervideo 10,007, may include a method permitting the object to fire an event message. Further, each object may be defined during hypervideo 10,007 authoring to have a method(s) that are event handler(s) for receiving event message(s) during hypervideo performance. An event handler is a method of an object that defines how the object responds to a received event message. Even if a hypervideo 10,007 may not communicate externally, objects of the hypervideo 10,007 objects may still communicate with one another by event messages.

The information retrieved from queries or logged information may be used to manipulate dynamic hypervideo performance. Event handlers may utilize variables (e.g. placeholders) corresponding to information resulting from queries, or the retrieval of logged information not performed by queries. An event handler may utilize such information to manipulate dynamic hypervideo performance. Both queries and event handlers may be executed by the dynamic hypervideo server 25,035.

Objects and Events

Numerous classes of objects may exist in a hypervideo 10,007. Many objects corresponding to each class may be instantiated. Each class of objects may have a unique set of event handlers associated with the class. Not all event handlers, which are methods, may be found in each class because some event

handlers are not relevant for certain classes. For each and every object in a hypervideo 10,007, events in an object type's event list may be selected for handling, either by using a handler specified for that specific event, or by firing the events to be handled in an external environment.

An exemplary list of object types and their associated events for one embodiment follows:

Object Type	*Event*	*Description*
Project	OnStart	The project started.
	OnEnd	The project ended normally.
	OnResize	The project window was resized.
	OnTerminate	The project ended due to external interference.
Hotspots	OnMouseClick	The hotspot was actuated by clicking the left mouse button.
	OnMouseOver	The cursor was passed over the hotspot.
	OnMouseRightClick	The hotspot was actuated by clicking on the right mouse button.
	OnStart	The hotspot appeared for the first time.
	OnEnd	The hotspot no longer exists in the hypervideo.
Hotframes	OnStart	The hotframes's frame display started.
	OnEnd	The hotframe's frame display ended.

Media	OnStart	The media started playing.
	OnPause	The media ended playing normally.
	OnStop	The media ended playing due to external interference.
LinkLevel	OnStart	The link (target) level started playing.
	OnEnd	The link (target)level ended playing.
	OnTerminate	The link (target) level ended playing due to external interference.
Link	OnStart	The link (target) started playing.
	OnEnd	The link (target)ended playing.
	OnTerminate	The link (target) ended playing due to external interference.
Frame	OnStart	The frame display started.
	OnEnd	The frame display ended.

When an event message, corresponding to an event, is fired from the client 25,003 to, for example, to the dynamic hypervideo server 25,035, the event message may include arguments, or parameters that indicate the object that has fired the event message. An argument may include a string specifying the name of the object.

Each object may have a Properties sheet 34,001, illustrated in Figure 34, that includes an Events page 34,003. In the Events page, the events 34,005 whose messages may be fired by the object may be specified (e.g. selected, by checking a box, from all events potentially fired by the corresponding object). From the Events page 34,003, event handlers, corresponding to the specified fired events may also be accessed by actuating (e.g. clicking) a Handler button 34,007, proximate to each specified event.

The Events page 34,003 may also include a Select All button 34,009, which when actuated, selects all events potentially fired by the corresponding object. The Events page 34,003 may also include a Set As Default button 34,011 which makes the selected events in the currently displayed Events page 34,003 the default configuration for the Events pages 34,003 for other objects of the same type.

In one embodiment, the Events page for an object is located in the properties sheet for that object. For example, the Events pages for the following objects are located in the co-listed properties pages, or as otherwise described below:

Project object- In the Project Setting Properties sheet.

Hotspot object - In the Hotspot Properties sheet.

Hotframes object - In the Hotframe Properties sheet.

Media object - In the Media Properties sheet.

Target Level object - In the links, or target, page in a Hotspot Properties sheet. Each level tab, or page, may include an Events button which may be actuated to display, or pop-up the corresponding Link Level object events sheet.

Target, or Link, Object - In the Media link properties sheet.

Frame Object - The Workshop toolbar may include an Events button which will display, or pop-up, the Frame object events page for the current frame.

Commands

Hypervideo application commands may enable the performance of a hypervideo 10,007 to be manipulated. The commands may be issued by

application(s) 25,038 or by the player 25,039, for example in response to the occurrence of an event. The commands may, for example, manipulate the interactive features of the hypervideo 10,007 by enabling or disabling hotspots 8003, manipulate the properties of different objects in the hypervideo 10,007, or launching a target 7003. Commands may include parameters which identify the action to be performed and the object to which the action is related. In one embodiment, the commands that may be submitted to a hypervideo 10,007 include:

Commands	***Parameters***	***Description***
Play()		
Pause()		
Close()		
EnableHotspot()	1. Boolean Enable Switch 2. String with Media Name 3. String with Hotspot Name	Enable or disable the specified hotspot in the specified media.
ActivateMedia()	1. String with Media Name	
ActivateLink()	1. String with Link (target) Name	Launch the specified Link (target) level. This command will launch only the specified level and not the entire set of link (target) levels associated with a media object.
ActivateHotspot()	1. String with Media Name	
SetProjectProperties()		
SetHotspotProperties()		

SetFrameProperties()		
SetMediaProperties()		
SetLinkProperties()		
SetLinkLevelProperties()		

These commands may be supported by the dynamic hypervideo server 25,035 and player 25,039.

Event Handlers

An event handler is a script associated with a specific event message that may be received by a specific object. Event handlers enable relatively inexperienced hypervideo 10,007 authors to create relatively complex hypervideos 10,007, for example using substantially all supported events and commands, and to permit communications between application programs 25,038 and other components of the hypervideo system 25,000.

General

In one embodiment, each object in a hypervideo 10,007 may have a unique event handler associated with each event message that the object may fire. Each event handler also may access parameters, for example object name, of the object that fired the event. These object parameters may be needed by the event handler to process the received event message.

In one embodiment, event handlers may be written in a simple scripting language, having the following form:

On Event {Condition}Do Command

Event handlers 26,203 may be viewed in the Handlers Notebook window 26,201, illustrated in Figures 26 and 35, in the authoring tool GUI 26,001, or through an Events page. In the Handlers Notebook window 26,201, different pages 26,209 displaying different event handlers 26,203 (each associated with an unique event message that may be fired by an unique object) that may be reviewed, edited or

created. A page 26,209 from the Handlers Notebook window 26,201 may also be displayed in the Events page, discussed above.

Handlers Notebook Window

The Handlers Notebook window 26,201 may be hidden in the authoring tool GUI 26,001. In one embodiment, the Handlers Notebook window 26,201 includes pages each of which includes a unique event handler 26,203 definition. Blank pages may be opened, and new handlers may be authored therein. New handlers created through an object's Events page are automatically added to new pages in the Handlers Notebook window 26,201.

In one embodiment, a particular page 26,209 which may contain a particular event handler 26,203 may be accessed in one of two ways. First, regardless as to whether an event handler is defined in it or not, a page 26,209 may be selected with specific event handler 26,203 parameters (e.g. object type and name, and the specific event) from combo boxes 26,205 in the Handlers Notebook window 26,201. If the selected page includes a predefined event handler, the event handler 26,203 may be displayed in the page 26,205 in an area 26,211 for displaying script. If the selected page does not include a predefined event handler, the script display area 26,211 may be blank. Pre-defined handlers on pages may be selected by manipulating (e.g. clicking) the Next and Previous Handler buttons 26,213. A new event may be created, or a pre-existing event edited in the script wizard, described below, which can be launched by actuating (e.g. click) the script wizard button 26,217.

Handler's Scripting Language

In one embodiment, the script associated with an event handler 26,203 may be defined in a scripting wizard. In one embodiment, the script area 26,211 in the Handler notebook, may be used to conveniently present handlers described in script form to users who are comfortable with the concept of scripting. Other users may review the event handler script to determine whether a handler has been created, and the complexity of the event handler. In this embodiment, any

modifications made to the event handler may be performed with the script wizard, in addition to altering the text in the text editing window.

In another embodiment, a more complicated and powerful scripting language (e.g. VisualBasic or JavaScript) may be used to permit experienced script programmers to integrate the application(s) 25,038 with other hypervideo system 25,000 components using events and commands.

The structure of one embodiment of an event handler follows:

At ObjectType ObjectName

On EventName

{If (QueryCondition1 | LoggingCondition1) }Command1

{If (QueryCondition2 | LoggingCondition2) }Command2

End

For Example:

At Hotspot Mosque

OnMouseClick

EnableHotspot (FALSE, Jerusalem, Church)

If (Query001 equal 10) ActivateLink (LinkMuslim001)

End

Note that data from a query or a log file 25,049 may define, at least in part, a logical expression, or conditional term, of an event handler 26,203.

Each event handler 26,203 may contain an unlimited number of commands, some of which have been previously discussed. Commands may be used to manipulate hypervideo 10,007 parameters. Each command may be placed on a separate line. Each command may be accompanied with a logical expression. The logical expression may include a placeholder, or variable, associated with a query or logged information. The command may be implemented only if the logical expression is satisfied. The logical expression may be a simple Boolean expression which compares the value of a first variable to the value of a second variable or constant.

Script Wizard

The Script Wizard 36,000, whose window 36,001 is illustrated in Figure 36, is used to generate the script for event handlers 26,203. The Script Wizard 36,000 permits an event handler 26,203 to be visually programmed without manual typing (e.g. code is generated based upon user selections in the Script Wizard 36,000). The Script Wizard window 36,001 may be launched from the Handlers Notebook window 26,201, by actuating (e.g. clicking) the Script Wizard button 26,217.

In one embodiment, scripts may be defined with the Script Wizard in two steps. First, commands and associated parameters may be defined. Second, conditional terms may be defined to manipulate hypervideo 10,007 parameters, described above. When a command is defined, the user can either choose the desired command from the combo-box, or click the "Alphabetical Command List" button or "Topical Command List" button, and choose the command from a pop-up menu. Upon command definition, a dialog may be displayed (e.g. popped up). Parameters associated with the command may be defined in the dialog. The dialog may also be displayed, permitting the parameters to entered or edited, by actuating (e.g. clicking) the Modify Parameters button 36,003 of the Script Wizard window 36,001. The defined parameter may be displayed in the parameters line 36,005 of the Script Wizard window 36,001.

Logical expressions may be added to the command (so that the command may be performed only if the logical expression is fulfilled) upon selecting (e.g. checking) the Add Expression box 36,007. Operands, including variables, and operators for the conditional terms may then be selected. The operators may be selected from combo box having a list of operators (including equal, greater than, etc.). To select an operand, a button may be actuated to display menu 37,001, illustrated in Figure 37, which contains a constant 37,003, a query 37,005 (from the Query Warehouse window 26,072), or a logging variable 37,007. Selected parameters are displayed in the appropriate field, and can be modified at any time in the Script Wizard window 36,001. Definition of the script in the Script Wizard window is completed by actuating (e.g. clicking) the OK button 37,009. Then, the script can be viewed in the Handlers Notebook Window 26,201.

Multiple sets of commands and logical expressions may be created using a multi-level approach, similar to the link, or target, levels described above. The commands sets may be manipulated, for example by dragging and dropping the commands to pages, or tabs, corresponding to the different command set levels.

Defining Default Events Settings and Event Handlers

Default settings may be defined for each object type. The default settings may include the default events configuration for each object type and default handlers for each event associated with a specific object type. Upon selecting the Set Default Settings option in the Settings menu in the main tool and menu bars, a Default Settings sheet 38,001, illustrated in Figure 38, may be displayed (e.g. popped up). In the Default Settings sheet, each object type may have an Events page (e.g. Project Events page 38,003, Hotspot Events page 38,005, Media Events page 38,007, Link Events page 38,009, Frame Events page 38,011, etc.) which may be identical to the Events page that each object of that type has in its corresponding properties page. The Events Settings and handlers defined in these Events pages are the default settings for the corresponding object types. Thus, any object, of an object type, created after the default settings have been defined may have the same selection of events, and corresponding event handlers, unless otherwise changed.

Logging

The hypervideo system 25,000 may permit hypervideo 10,007 activities to be logged during hypervideo 10,007 performance. In one embodiment, an author of a hypervideo 10,007 may specify, during hypervideo 10,007 authoring, which viewer interactions (or other hypervideo 10,007 activities) and corresponding information are to be logged. The logged information may be used to manipulate the performance of the hypervideo 10,007.

General

In one embodiment, the logging of hypervideo 10,007 activities and corresponding information may be performed in relation to objects defined in the

hypervideo 10,007. For each object in the hypervideo 10,007, corresponding actions to be logged may be defined. In one embodiment, media files may be objects, and thus the activities (e.g. starting and stopping) of corresponding media files may be logged. This approach may be analogous to the approach used for defining objects's events.

In one embodiment, each object may be associated with one of three logging levels. Successive logging levels log more extensive information that previous types, and may include:

a. Count: Logs the number of times the object was acted upon (e.g. actuated, performed, stopped, etc. For example, the number of times a hotspot was actuated may be logged.

b. Duration: Beside logging the information associated with the Count logging level, the Duration level also logs the duration of an object's activity (e.g. performance time of a media file) may be logged.

c. Timing: Besides logging the information associated with the Duration logging level, the Timing level logs the time when the object activities occurred (e.g. when the media file performance started and stopped, or when a hotspot 8003 was actuated).

A list of the objects for which related activities may be logged, and the types of activities that may be logged for those objects follows:

Object Type	**Logging Types Available**	**Description**
Project	Count	Number of times a hypervideo 10,007 was performed, or activated, and the identifications (IDs) of the viewers who activated the hypervideo 10,007.

Object Type	Logging Types Available	Description
	Duration	The total time of each performance of the hypervideo 10,007.
	Timing	The start and stop times of each performance of the hypervideo 10,007.
Hotspots	Mouse Click : Count	Number of times a hotspot was actuated (e.g. clicked) during a performance.
	Mouse Click : Timing	Times when the hotspot actuations occurred.
	Mouse Over : Count	Number of times a pointer was positioned over a hotspot.
	Mouse Over : Duration	Total time that a pointer was positioned over a hotspot.
Media	Count	Number of times a media was performed, regardless of associated link (target), during a performance of a hypervideo 10,007.
	Duration	Number of frames of the media displayed (e.g. during each media performance).

Object Type	Logging Types Available	Description
	Timing	Start and stop times (relative to hypervideo 10,007 start time) of the media.
Link Level	Count	Number of times that a link (target) level was activated during the performance of a hypervideo 10,007.
	Duration	Activation time duration of a level.
Link	Count	Number of times a target was executed during the performance of a hypervideo 10,007.
	Duration	Time duration of target execution.
Frame	Count	The number of times a frame was displayed.

Logging Design

In one embodiment, a logging tier, or level, may be selected for each object. Each object may have a Logging page 39,001, illustrated in Figure 39, in its properties sheet. In the Logging page 39,001, the logging tier may be specified.

In one embodiment, logging pages 39,001 may be accessed, as described below, for the following object types:

Project object- In the Project Setting Properties sheet.

Hotspot object - In the Hotspot Properties sheet.

Hotframes object - In the Hotframe Properties sheet.

Media object - In the Media Properties sheet.

Target, or Link, Level object - In the Target, or links, page in the Hotspot Properties sheet, each Target Level page will include an Events button which when actuated displays (e.g. pops-up) the Target Level Object Events page from which the Logging page may be accessed.

Link Object - In the Media link (target) properties page.

Frame Object - The Workshop toolbar may include an Events button which may display (e.g. pop-up) a Frame object events page for the current frame from which the Logging page may be accessed.

Defining Default Logging Setting

Default logging settings may be defined in a manner similar to the way that default settings for the events are defined as described above. Thus, in one embodiment, the Logging page 39,001 may include a Set As Default button 39,002, which may be actuated to make the logging data in the Logging page 39,001 the default data for Logging pages 39,001 for other objects of the same type (e.g. any object of the same type that is later created will have the same logging data, unless otherwise defined).

Using Logging Information in a Dynamic Hypervideo

In one embodiment, as described above, logged information may be used to manipulate hypervideo 10,007 parameters during the performance of the hypervideo 10,007, by embedding the logged information (e.g. parameters) inside logical expressions as operands, or variables, or placeholders. In one embodiment, the operands (e.g. of the logical expressions) for logged information may be dynamically updated in response to viewer interactions during hypervideo 10,007 performance. To reference these variables, each variable corresponding to a logged parameter may be automatically assigned a logical name by the

hypervideo system 25,000. In one embodiment, the logical names may have the following format :

ObjectType_ObjectName_LoggedFeature

An example of a logical name may be: Hotspot_Hotspot001_Count or: Media_Jerusalem_DurationLast. A logged parameter may be selected as a variable in a logical expression by selecting the logging option in the Query Wizard when prompted to choose an operand. Then, a variable corresponding to a desired logged parameter may be selected from a combo box.

Logging during Hypervideo Performance

A new record of logged information may be created upon the occurrence of an event (e.g. the actuation of a hotspot 8003). Each record of logged information includes the information, about a specific event, specified by a hypervideo developer. In one embodiment, logged information record formats may include: MouseClick(user_id, project_name, hotspot_name) and MediaPlay(user_id, project_name, media_name, duration). In one embodiment, the logged information may be stored, by the dynamic hypervideo server 25,035, in a file (e.g. log file 25,049 or database), in the form of list, having a proprietary format. In another embodiment, the hypervideo dynamic server 25,035 may facilitate exporting (e.g. with ODBC drivers) and storing logged information in a database 25,037, such as a SQL database. Logged information in a database 25,037 may be viewed and analyzed with the logging tools 25,047, or with other database tools.

The logged information may be stored in a client 25,003a so as not to diminish the bandwidth available on the network 25,050a for video transmission. The logged information may be periodically transferred to the server 25,005 for storage during hypervideo 10,007 performance, or upon hypervideo 10,007 termination.

Logging Analysis

The hypervideo system 25,000 may include logging tools 25,047 which may permit viewing the logged information associated with the performance of a hypervideo 10,007. The logging tools 25,047 may support retrieving logged information, including different sets of logged information (e.g. different sets corresponding to different performances of the same hypervideo 10,007), to permit statistically analyzing the sets of logged information. The logging tools 25,047 may read and analyze logged information stored in different logging file formats (e.g. proprietary file format or conventional database format).

Hypervideo Application Integration Requirements

The authoring tool may permit an author to access features and functionality in the hypervideo system 25,000. The player 25,039 may include a modified version of the run-time module 1001. The player 25,039, which may include a run-time module 1001, may be integrated with the relevant site building tools used to build web sites and the resulting web sites. Thus, the player 25,039, may be included in a web browser as a standard control (e.g. ActiveX control). The player may have to implement interfaces of relevant API architectures (e.g. COM) to communicate with the web browser.

2.5 Object Oriented Implementation of Hypervideo System

Authoring Tool

As described above, the authoring tool 1001 facilitates the creation of hypervideos 10,007. One embodiment of a procedure for creating a hypervideo 10,007, depicted in Figure 40, is described below.

Author-selected media files may be imported into the Media Warehouse window 10,003 (step 40,001). Media elements 1690, that are to be base target(s) 1109 for the hypervideo 10,007 may be selected using the project settings dialog (step 40,002). A media element 1690 may be dragged into the Workshop window 8001 (Step 40,003). A media object may be defined in the media associated with the media element 1690 (step 40,004). In one embodiment, the

media object is a hotspot, for example, which may be defined in a frame of video over a region of interest. The media object may be defined in the frame in which the region of interest first appears in the video. Then, the defined media object corresponding to the region of interest is tracked in successive video frames (steps 40,005a,b,c). The media object may be tracked automatically 40,005a, manually (or semi-automatically) 40,005b, or by interpolation 40,005c, which are described above. In one embodiment, for example, target(s) 7003 may be linked to the media object using the Target page 7001 of the Hotspot Properties sheet 7101 (step 40,006). The shape, or type, of the over and click cursors, described above, associated with the media object may be defined (step 40,007) in the Hotspot Properties sheet. A marker 1155 to identify the media object during hypervideo 10,007 performance may also be defined (step 40,008). Steps 40,003 through 40,008 may be repeated to define more media objects in the same or different media. Upon completing media object definition, the hypervideo 10,007 may be saved (step 40,009). In one embodiment, the hypervideo 10,007 may be previewed (step 40,010) in the authoring tool 1001 or in the run-time module 1001. The hypervideo 10,007 may then be published so that is accessible for use by run-time modules over computer networks (step 40,011).

In one embodiment, the authoring tool 1001 may be used in conjunction with other video editing tools, such as Adobe Premiere (Adobe Systems, Inc., San Jose, CA). In another embodiment, data defining a hypervideo 10,007 may be created and saved in a project file 1670 (e.g. .OBV file) by the authoring tool 1001 using the following libraries and dynamic linked libraries (DLLs):

a) Object database 1127 may be a library configured to manage the data of the Project File 1670.

b) MediaDisplay is a library that provides the services necessary to display or execute various media.

c) FrameManager is a library that provides the services needed for extracting frames from within media files.

d)	Tracker	is a library that tracks the geometry, size and position of a media object in successive video frames.
e)	BitMaster	is a DLL that performs the mathematical calculations and data processing necessary for the Tracker operation.
f)	ShapeIterator	is library that iterates the geometric shape of a hotspot into scan-lines for use by the Tracker as well as the marking 1155 mechanism of the run-time module.
g)	MCIVA	is a run-time module DLL that performs the preview function when the authoring tool 1001 is used.

In one embodiment, the authoring tool 1001 may be a Microsoft Foundation Class (MFC) application. In another embodiment, the authoring tool 1001 may also be an Multiple Document Interface (MDI) application. As an MDI, the authoring tool 1001 may contain multiple templates (e.g. windows), including the Media Warehouse window 10,003, the Workshop window 8001, and the Preview window 8101. Each template may include document, view, and frame classes. This structure of windows and classes is known to persons skilled in the art of Microsoft Windows computer programming. The templates and their classes are described below:

Media Warehouse Window

In one embodiment, the Media Warehouse window 10,003 may be implemented with the logical class relationships illustrated in Figure 41. Within the Media Warehouse window 10,003 are the document, view, and frame classes, described below:

document: may be described by class EObviousDoc 41,003 which may include a parameter for a pointer to an EProject object 41,002.

The EObviousDoc 41,003 class may include methods for saving hypervideo 10,007 data to and accessing hypervideo 10,007 data from a project file 1670 (e.g. respectively OnOpenDocument() and OnSaveDocument() methods). In one embodiment, the EObviousDoc 41,003 may include functions similar to those available within EProject 41,002, such as *Load()* and *Store()* for respectively accessing and saving hypervideo 10,007 data.

In one embodiment, prior to performing saving function, the hypervideo 10,007 data may be compressed, or encrypted. Because the hypervideo 10,007 data may be compressed, the document may not be serialized in a standard fashion. Serialization of the hypervideo 10,007 data may be avoided, for example, by overriding the EDocument=sOnOpenDocument() and OnSaveDocument() methods in the EDocument class.

view: may be described by class EObviousView 41,005. The EObviousView class 41,005 may include methods for displaying the Media Warehouse window 10,003. The media element 4007 may be inserted into the view after the project is read and the view may have been initialized. Initializing a view class object is known to persons skilled in the art of Windows Programming. If the class is unable to locate a media file corresponding to a media element 4007, a help window is opened. The window prompts the author to enter data to assist in locating the corresponding media file. Upon failing to locate a media file, the corresponding media element is deleted from the hypervideo. A similar technique is used for cursors 7111, 7113 in a hypervideo 10,007.

A media display function in the EObviousView class 41,005 may facilitate the display of thumbnails 4007 corresponding to the media associated with the Media Warehouse window 10,003. The EObviousView class 41,005 may include parameters for storing an array of thumbnails 4007. The stored thumbnails 4007 may be used by other classes in the authoring tool 1001.

In one embodiment, the EObviousView class 41,005 may include other methods. In another embodiment, the EObviousView class 41,005 may include methods for accessing media properties by using methods from the EMedia class which is described in the Object database 1127 section below. For example, the EObviousView class 41,005 may open a media menu. Furthermore, The EObviousView class 41,005 may permit the hypervideo 10,007 to be previewed by using an application program interface (API) of the run-time module 1101, which may be a DLL.

frame: may be described by the EObviousFrame object 41,004. The EobviousFrame object 41,004 creates and displays a control bar, and menu commands, of the Media Warehouse window 10,003.

Media is displayed in the Media Warehouse window 10,003 using methods in the EMediaList class 41,006 . The EMediaList class 41,006 may be derived from the Microsoft (TM) Foundation Class (MFC) CListCtrl which is an object oriented programming class known to persons skilled in the art. In one embodiment, in addition to displaying media, the EMediaList class 41,006 may include additional methods, including those that:

a) Control cursor appearance and configuration. For example, in one embodiment, the EMediaList class 41,006 may include a method that permits modifying the cursor type based upon the image behind the

pointer. In one embodiment, for example, a "hand-shaped" cursor may be displayed when the pointer is positioned over a media. An "arrow" cursor may be displayed when the pointer is not over media.

b) Open the Media Properties Sheet 5003. In one embodiment, the Media Properties sheet 5003 may be opened when a switch is actuated (e.g. a mouse button is double-clicked) while the pointer is positioned over the corresponding media element thumbnail 4007. In one embodiment, actuation of the ENTER key on the keyboard may open the Media Properties sheet 5003 when the pointer is positioned over the corresponding media thumbnail 4007 and a switch was actuated (e.g. a mouse button is clicked).

c) Manage the display and content of various Windows messages (which are known to persons skilled in the art of Windows Programming) associated with functions. In one embodiment, the EMediaList 41,006 class may include a method that generates and displays context-sensitive "drag and drop" messages for functions such as linking a media object to a target 7003. In one embodiment, a portion of the Media Warehouse window 10,003 may display a hotspot menu. This hotspot menu may be derived from a general purpose Menu class in the MFC library and is known in the art as a pop-up menu. In another embodiment, the hotspot menu is opened when a switch is actuated (e.g. the alternate mouse button is clicked) while the pointer is positioned over the corresponding media element thumbnail 4007. In one embodiment, each item in the hotspot menu may be the name of a hotspot 8003. In another embodiment, the menu items may include thumbnails 4007 of the corresponding hotspots 8003. In one embodiment, upon selecting a hotspot menu item, targets 7003 of the selected hotspot 8003 are displayed. Targets 7003 are displayed by popping up another pop-up menu with a list the selected hotspot's 8003 targets 7003. In this Targets 7003 pop-up menu, the targets 7003 may be identified by name. In another embodiment, a

thumbnail 4007 of the media element associated with the target 7003 may be displayed.

Workshop Window

In one embodiment, a hypervideo 10,007 may be created in the Workshop window 8001. In a Microsoft Windows (TM) environment, within the Workshop window 8001 are the document, view, and frame classes, described below:

document: may be described by class EClipDoc 42,003, depicted in Figure 42. In one embodiment, the class EClipDoc 42,003 may include parameters for storing data associated with the current frame being displayed in the Workshop window 8001, as well as a list of objects instantiated from a class EDrawObj 42,002. In another embodiment, the class EClipDoc 42,003 may also include parameters for storing hotspot tracking 1029 and other data. In one embodiment, the class EClipDoc 42,003 may include methods for displaying a video frame or an image in the Workshop window 8001, and for providing hotspots 8003 associated with (e.g. on top of) the video frame or image displayed in the Workshop window 8001.

view: may be described by class EClipView 43,002, depicted in Figure 43, may include methods, including "Draw Rectangle", "Cut" and "Undo". In one embodiment, the class EClipView 43,00 also may include methods for creating, editing, and manipulating hotspots 8003. The class EClipView 43,002 further may include a method for displaying a window in which a video frame or an image is displayed. In one embodiment, the window is located inside the EClipView window, and in another embodiment, the window may be smaller than the screen area and projected as a function of the size of the media and author-defined specifications. These

specifications, for may be defined in the Options Settings menu item in the main menu bar of the authoring tool 1001.

frame: may be described by class EClipFrameWnd 44,002, depicted in Figure 44, and may include, in one embodiment, methods for creating and manipulating the workshop toolbar, the hotspot toolbar 9001, and the workshop control bar illustrated with buttons in Figure 8b, and corresponding functions (e.g. tool functions).

The following tool functions may be available from within the Workshop window 8001:

Draw Objects

In one embodiment, the class EDrawObj 42,002 and its derived classes may include methods for managing hotspot 8003 data associated with a displayed frame. In another embodiment, for a frame-object, instantiated from class EFrameObject 42,005, corresponding to the currently displayed video frame, the authoring tool 1001 may create an object instantiated from class EDrawObj 42,002. In addition to including a parameter for storing a pointer to a frame-object, class EDrawObj 42,002 may include methods for drawing, resizing, and moving a hotspot 8003.

Classes which may be derived from class EDrawObj 42,002 include:

EDrawRect 42,006	which may include methods for drawing non-polygon-shaped hotspots
EDrawPoly 42,007	which may include methods for drawing for polygon-shaped hotspots.

In one embodiment, an object is instantiated from the class EDrawObj 42,002 when a new hotspot 8003 is drawn. In another embodiment, when creating the EDrawObj 42,002 object a new EClipObject 43,011 is instantiated for the video, or clip, (e.g. in an object database 1127) as well as a new EFrameObject 48,021 with the appropriate geometry. In one embodiment, the hotspot name may be assigned automatically and may feature a consecutively numbered suffix. The object oriented representation of a hypervideo 10,007 project is further described below. In one embodiment, the EDrawObj may handle the visual representation of a hotspot 8003 while the EClipObject 48,026 and EFrameObject 48,021 objects handle the object database 1127 representation of a hotspot 8003.

Draw Tools

In one embodiment, the draw-tools are the classes that perform operations in the workshop 8001. The author selects a draw-tool by choosing from the options displayed on the hotspot tools toolbar 9001. The current draw-tool may be stored in the static member *EDrawTool::c_drawShape* and, in one embodiment, options include: *selection*, *set_target*, *rect*, *triangle*, *ellipse* and *poly*. In one embodiment, the draw tools are also responsible for setting the cursor to the appropriate configuration, namely, pencil, arrow or set target cursors.

The draw-tool classes (derived from EDrawTool 43,003) may include:

ESelectTool 43,006	In one embodiment, this may be the default tool. In one embodiment, the author can move the handle by actuating a switch (e.g. clicking on the left mouse button) while the pointer is positioned on the handle of a selected hotspot 8003. In an object oriented shape drawing tool, the may handle are areas often positioned at vertices of a shape, which an author can move in a two dimensional space and by moving the handle, and resizing the drawn shape. In one embodiment, the hotspot 8003 may be selected if the cursor was positioned within the frame of a defined hotspot 8003. In one embodiment, multiple hotspot 8003 selections may be made by depressing the shift key while manipulating the mouse and its buttons. In one embodiment, all hotspots located within a rectangular region may be selected using a net selection. A net selection is a term known to persons skilled in the art. A net selection may begin

when drawing of a rectangle by depressing and holding down the left mouse button. In one embodiment, the hotspot(s) selected are stored in the global variable selectMode. The variable may reflect a value of *none* (meaning no selection was made), *netSelect*, *move* (meaning that a hotspot 8003 may have been dragged to a new location), *size* (a hotspot is being resized) and *to_be_created* (a hotspot 8003 is being created - wherein this mode is set by draw tools 43,003 that creates hotspots 8003).

ERectTool 43,004

In one embodiment, this tool may be used for drawing non-polygon-shaped hotspots 8003. One method of defining non-polygon-shaped hotspots 8003 will now be described. In one embodiment, the hotspot may be defined by a bounding rectangle. Clicking the left mouse button may define the first corner (for example, the upper-left corner) and releasing the left mouse button may define the second corner (for example, at the lower right) of the bounding rectangle. When the left mouse button is initially clicked, the global variable *selectMode* is set to *to_be_created.* In one embodiment, the hotspot 8003 is created upon release of the mouse button and further movement of the pointer. In one embodiment, an EDrawRect 42,006 object, and corresponding EClipObject 48,026 and EFrameObject 48,021 objects are instantiated in the object database 1127, described in Figure 48, when a hotspot 8003 is created. In one embodiment, *selectMode* may be set to *size*

during hotspot 8003 definition. In one embodiment, pointer movement after hotspot creation may be handled as a drag of the handle wherein ESelectTool=s 43,006 method is called as described above. In one embodiment, upon release of the mouse button, the physical size of the rectangle is checked, and if the physical size is determined to be too small, the corresponding hotspot 8003 objects are deleted.

EPolyTool 43,005 In one embodiment, this tool may be used for drawing polygon-shaped hotspots 8003. One method for defining polygon-shaped hotspots 8003 will now be described. In one embodiment, clicking a mouse button may define a new EDrawPoly 42,007, which, as described above, creates the corresponding EClipObject 48,026 and EFrameObject 48,021 objects in the object database 1127. In one embodiment, hotspot 8003 definition includes clicking the mouse button when the pointer is on the first vertex and dragging the pointer to a second vertex. In one embodiment, additional vertices of the polygon are defined by clicking and dragging the pointer location, as describe above. Double-clicking the left mouse button causes the last vertex to be deleted. The hotspot 8003 is deleted if it does not satisfy a requirement for minimum vertices.

ESetTargetTool 43,008 In one embodiment, this tool may be used for linking a target 7003 to a selected hotspot 8003. A method for linking a target 7003 to a hotspot 8003 will now be

described. In one embodiment, the hotspot 8003 may be selected by the author making a first mouse click while the pointer is positioned on the hotspot 8003 in the Workshop Window 8001. In one embodiment, movement of the mouse may begin the drag and drop operation. The pointer may be dragged over to the Media Warehouse window 10,003 and the pointer positioned over a media element thumbnail 4007. In one embodiment, the media element thumbnail 4007 may be automatically selected and by releasing the left mouse button. Then, the drag and drop operation is completed. Also, the media element to be linked as a target 7003, to the previously selected hotspot 8003, is defined.

After using any one of the draw-tools, the current tool reverts back to *selection.*

Undo In one embodiment, the undo mechanism may allow the author to undo operations performed in the workshop window 8001. In one embodiment, three types of operations may be undone: (1) moving a hotspot 8003 in the frame of the video or image, (2) deleting a hotspot 8003 and (3) changing the Z-Order (layering) of a hotspot 8003. The Z-Order is a term known to persons skilled in the art and determines how hotspots 8003 are placed upon one another when two or more hotspots 8003 overlap a region in a frame of video or image. In one embodiment, the definition of a hotspot 8003 may also be undone. In one embodiment, hotspot 8003 interpolation may also be undone.

In one embodiment, the EClipView object 43,002 may store an array of EUndoCommand 43,017 objects, wherein each array stores the data of one operation performed in the workshop window 8001. In one embodiment, the undo function is multi-level, meaning that more than one operation may be undone. In one embodiment, memory considerations may make it advisable to limit the number of undoing operations available. The undo array may be deleted every time a new media is brought into the Workshop Window 8001. In one embodiment, the undo command may be performed regardless of the frame in which the operation was performed. In otherwords, the undo command may be operational for all frames, including those frames not currently selected. In one embodiment, the performing the undo operation may result in the selected frame being the frame in which the undo operation was performed.

In one embodiment, an operation performed on a number of selected hotspots 8003, is treated as a single operation. Performing the undo command to undo that operation affects all previously selected hotspots 8003.

In one embodiment, a redo array may be created as part of EUndoCommand 43,017 objects which is stored in the clip view. Upon performing a redo command, associated with the redo array, undone operations may be reperformed. In one embodiment, the redo array may be deleted upon bringing another media into the Workshop window 8001.

The following describes the three types of operations that may be undone, according to one embodiment:

EUndoObjectMove 43,018

In one embodiment, this function may store an array of the hotspots 8003 that were recently moved including the original geometry for each hotspot

8003. Upon performing the undo function, restores the original geometry of the hotspots 8003.

EUndoObjectDelete 43,020

In one embodiment, this function stores an array of the hotspots 8003 that were recently deleted. Because a hotspot 8003 may be deleted in multiple frames of the video, each hotspot 8003 may be associated with an array of stored geometries (for a specified range of video frames). Upon calling the function *Undo()*, these geometries may be restored. If a hotspot 8003 is removed (by using the *remove* option, or by deleting all its EFrameObject objects 48,021), the hotspot 8003 may be deleted from the object database 1127. In this case, the EClipObject 48,026 object which was removed from the object database 1127, may be stored with the undo data, and restored into the object database 1127 by the *Undo()* function.

EUndoObjectZOrder 43,019

In one embodiment, this function may store an array of the hotspots 8003 for which the Z-Order may have changed. The Z-order specifies the layering of hotspots 8003 and is described above. Because the Z-Order may only be changed for one hotspot at a time, this array consists of a single item. Since layering of hotspots 8003 across multiple video frames may be available, each hotspot 8003 may be associated with an array of stored indices corresponding to each video frame. When the U*ndo()* function is called the hotspot is restored to its original index in all the frames.

Preview Window

This window, according to one embodiment, performs a preview of a hypervideo 10,007 by actuating the run-time module 1101 and playing the hypervideo 10,007 in a window. In one embodiment, within the Preview window, are the document, view, and frame classes, described below:

document	the document may be described by class EClipDoc 42,003, like the workshop window
view	the view may be described by class EClipPreView
frame	the frame may be described by class EclipPreviewFrame

Dialogs

In one embodiment, operations may be performed with dialogs and property sheets. Dialogs and property sheets are known to persons skilled in the art of Windows (TM) programming. Each property sheet may be implemented by a class.

In the object oriented representation of the authoring tool GUI, the class associated with a property sheet is not aware of the classes associated with the pages of the property sheet. In one embodiment, the settings property sheet is aware of the classes associated with the pages of the property sheet. The pages are added to the property sheet by the class that created it (and not by the class of the property sheet). This technique may allow a particular property sheet to be used for a some multiple uses, wherein each use adds different property pages having different attributes. For example, in one embodiment, a single media property sheet class may be used for all media types and yet each media type may have different pages.

In one embodiment, the authoring tool 1001 may utilize the following dialogs and property sheets:

Media property sheet

The media property sheet 5003 illustrated in Figure 5 may be used to view and change information about a media element. The media property sheet 5003 may be created and managed by an EMediaProperties object.

In one embodiment, the media property sheet may include up to five pages wherein each property page is associated with a class. The pages in

the property sheet may be a function of the type of media the pages represent. For example, the five media property page classes may include:

1. EMPGeneralPage Property Page (illustrated in Figure 5)

This is the general page, which in one embodiment, exists for all media types (except may be for URL address media). In one embodiment, this page may contain general information about the media, including, at least, type 5019, subtype, and compression 5017. In one embodiment, the author may change the name of the media element with this page. Because the dialog remains the same for all media types, and yet the information displayed for each media type is different, the dialog may contain fields having general names including *header1, content1, header2, content*, etc.. When the dialog is opened, these fields may be filled with the appropriate data, for example, according to the media type.

2. EMPPreviewPage Property Page

In one embodiment, this page may enable the author to preview the media, including bitmap, text, video and sound media types. In one embodiment, the media may be displayed using a media-window 6001 (except for bitmap media, which may be displayed on the page).

3. EMPLinksPage Property Page 7011

In one embodiment, this page displays a list-control (i.e. MFC control known to persons skilled in the art) containing all the hotspots 8003, which are defined in the media. In one embodiment, using a media-display method, the hotspots 8003 are displayed as a thumbnail of the media 4007 in the first frame in which the hotspot is defined. Double clicking the mouse button

on a hotspot 8003 may open the property sheet associated with the hotspot 8003.

4. EMPDefaultsPage Property Page

In one embodiment, this page provides default cursors data for the video and bitmap.

5. EMPPublishPage Property Page

In one embodiment, this page allows an author to determine how the hypervideo media files may be published and retrieved for use in a hypervideo 10,007. This page may be applicable for all media types.

URL media which were created by the author (using the *Create new media* menu option) have only a General page (e.g. EMPURLGenPage object).

Hotspot properties sheet

A hotspot properties sheet 7101 may be created and managed by an **EObjectProperties** object. In one embodiment, the hotspot properties sheet 7101 may include three pages as follows:

1.	EOPGeneralPage	This is the general page.
2.	EOPLinksPage	This page is the target page and, in one embodiment, may contain a tab-control of target levels and a list-control of the targets for the current level. Both tab-control and list-control are Windows (TM) API controls that are known to persons skilled in the art of Windows (TM) programming. In one embodiment, drag and drop targets 7003 and drag and drop levels 7601 is

implemented using the below described Drag & Drop methods. In one embodiment, when the *add-new-target* icon is dragged onto a media element thumbnail 4007 in the media window 10,003, the media window 10,003 may open the Target properties sheet described in Figure 7e, and may automatically add the new target. The levels tab 7601 is an EdragableTab class object, which is a tab-control which may handle dragging and dropping of tabs. In one embodiment, changes made to the target properties sheet 7001 may be applied when the OK button of the target properties sheet 7001 is actuated. Addition, deletion of and reordering targets 7003 or levels 7601 may be applied when the OK button of the hotspot property sheet 5003 is actuated. In one embodiment, this functionality is implemented by duplicating a copy of the EClipObject 48,026 object from the object database 1127 and changing the target for this copy. When the OK button is actuated the target levels 48,015 may be copied to the original object in the object database 1127.

3. EOPVisualPage Property Page This is the marking 1155 page.

Target properties sheet

In one embodiment, the target properties sheet is handled by an ETargetProp object and may contain up to 3 pages: General, Frames/Bounds and Display.

In one embodiment, different media types have different property page classes, according to the type of the linked media. In one embodiment, the names of the classes follows the form:

ESTxyyyPage wherein x denotes the type of the target media, such as V for video, B for bitmap, etc., and wherein yyy denotes the page (Parms for *General* page, Place for *Display* page and Mark for *Frames/Bounds* pages). Using this naming convention, for example, ESTTPlacePage denotes Display page for Text target media properties. The General, Frames/Bounds and Display pages will now be described:

2. General page 7801: This page may handle general information about the link (target), and it applies to all types of media. This page is illustrated in Figure 7e.

2. Frames page (for video 1011) and *Bounds* page (for sound 1015) 7901: This page may be a media window with a media bar and enable the author to set the frame range of the target. This page is illustrated in Figure 7f.

3. *Display* page (for video, bitmap and text media) 7460: This page is illustrated in Figure 7g. In one embodiment, a thumbnail of target media is displayed over a thumbnail of the active media (in the first frame in which the hotspot 8003 is defined in the video), and the author may set its position and size. In one embodiment, for video target media, the media is drawn in its IN frame 7903 (as set in the *Frames* page). In one embodiment, to handle the resizing and relocating of the linked media, an EResizeObj object is used. In one embodiment, if the location is relative to the hotspot, the hotspot's geometry

is drawn on the active media and the linked media thumbnail 1013 is located relative to it. In another embodiment, if the location is relative to the mouse position, the hotspot's click-cursor 7113 (as set in the *General* page) is drawn in the center of the hotspot 8003, and the linked media location is relative to this point (in runtime the location will be relative to the actual point where the click was performed, not to the center of the hotspot 8003).

Project Settings sheet

In one embodiment, this property sheet may handle settings that are a part of the hypervideo 10,007 (e.g. an Eproject object 48,001) and are saved as part of the project file 1670. The settings of the project setting property sheet will be used when the project is run in the run-time module 1101.

In one embodiment, each project may have a base-object, which is the first EClipObject 48,026 object in the hypervideo 10,007. The targets 48,009 associated with an EClipObject object 48,026 may be activated when the project starts running. In one embodiment, the project settings property sheet is implemented as a hotspot property sheet 7101 of this base object, and an EObjectProperties object is used. In one embodiment, the targets page is the same as the hotspot properties Targets page 7001, since, it too, consists of defining targets to the base-object. In one embodiment, the Settings and Media Location pages of the project settings correspond to the General page 7801 and Marking page 1155 of the Target properties sheet 7703.

Settings Properties Sheet

The Settings Properties sheet may handle settings that are not a part of the hypervideo 10,007, but rather, a part of the authoring tool 1001 environment. These settings do not affect the operation of the hypervideo 10,007 when it is run in other environments.

In one embodiment, default hard-coded settings are set by EObviousDoc 41,003 when the authoring tool 1001 program is first run on a particular computer. In one embodiment, the author may change these settings in this property sheet. The author may choose to accept the settings in the Settings properties sheet by clicking the OK button. In one embodiment, the settings are operative for the current instance of the authoring tool 1001 program. In one embodiment, the next time the program operates, the settings are reset. By selecting the Set As Default button, the current settings in the settings property sheet are put in the Windows Registry, known to persons skilled in the art, and these settings will be the default. In one embodiment, clicking the Reset button sets the settings in the property sheet to the default ones (i.e. taken from the Windows Registry, if there are any, or the hard-coded ones, otherwise). This will have no effect unless you click the OK button afterwards.

The Setting properties sheet is handled by an ESettings object. In one embodiment, this class provides the buttons Set As Default and Reset at the bottom of the property sheet, dispensing with the need for the traditional Apply button which is a standard Windows Property Sheet button known to persons skilled in the art.

The property pages are handled by the following classes:

Tracking page	ESettingsTrackPage
Advanced page	ESettingsAdvancedPage
Workshop page	ESettingsWorkshopPage
Preview page	ESettingsPreviewPage
Magic Wand page	ESettingsMagicPage

Project View

In one embodiment, the project view 1007 may be implemented as a modal dialog. A modal dialog is a type of dialog window that is known to persons

skilled in the art. In one embodiment, operations performed by the author in the authoring

tool 1101 with the project view 1007 may change the contents of the other windows, for example, the Workshop window 8001 and the Media Warehouse window 10,003.

The class which implements the project view 1007 dialog may be EProjView. In one embodiment, The project view 1007 dialog may include three buttons (Large icons, Small icons, and Text mode) and a tree-control. A tree control is a Windows API control that is known to persons skilled in the art. In one embodiment, the three buttons are part of a toolbar. This method of displaying elements in a tree-control in one of different modes (e.g. Large icons, Small icons and Text mode) is frequently used by Windows programmers and is known to persons skilled in the art.

In one embodiment, each node, or icon 11,003 in the tree control may be either a target 7003 or a hotspot 8003. In one embodiment, nodes 11,003 located in odd levels of the tree are Targets 7003 and nodes 11,003 located in even levels are hotspots 8003. The thumbnail icons 11,003 of the nodes are created using the MediaDisplay DLL, that is, hotspot 8003 thumbnails 11,003 with the hotspot 8003 drawn on the nodes. In one embodiment, target 7003 thumbnails are depicted in a reduced size and may include a frame (according to the type of the media). In small icons mode, each type of media may have a different icon 11,003, and all hotspots 8003 may have the same icon 11,003.

A tree may be characterized as infinite where a media links to itself, or to another media that ultimately links to itself. To avoid the problems associated with infinitive trees, in one embodiment, the following rule is used:

if the hotshot 8003 node (A) 11,003 DOES NOT have an uncle (B) with the same hotshot, then descendants of a hotshot node (A) are displayed. An uncle is defined as a node (B) 11,003 that has an ancestor of this node

(A) or a brother of one of its ancestors. In such case, the hotspot node (A) may have no children in the tree. The father-child terminology for describing nodes relationships in a tree data structure is known to persons skilled in the art.

This rule guarantees that if the children of a certain hotspot node 11,003 is discarded, then another node with this hotspot is currently visible, although not necessarily on the screen, because its father is expanded.

Locate Media

In one embodiment, when a project is opened, and if either the media or cursor file can not be found, then the Locate Media dialog window is opened to allow the author to help locate the missing file. This dialog is of class ELocateMedia. In one embodiment, this is a derived class of CFileDialog, which may handle the correct extensions for the files, and allows some additional options. In one embodiment, the main option is the Skip All option (Ctrl+Skip button).

The Application

In addition to the document templates and the dialogs, the MFC application mechanism maintains general information concerning the authoring tool 1001 application. This is an EObviousApp object (which is a CWinApp derived class. The CWinApp class is a standard MFC known to persons skilled in the art of Windows Programming). In one embodiment, another general object for the application is the main frame of the application, which is an EAppFrame object (CMDIFrameWnd derived class. The CMDIFrameWnd class is a standard MFC class known to persons skilled in the art of Windows Programming). In one embodiment, these objects can be accessed at any time by every class in the application using the global functions *AfxGetApp* (to get a pointer to the application) and *AfxGetMainWnd* (to get the a pointer to the main frame).

Hence, general data for the application, which is located in those objects, is available globally.

EObviousApp

This class may handle the initialization of the application in the function *InitInstance*. In one embodiment, this class performs numerous functions, including creating and displaying the document-templates, reading the registered data from the Windows Registry, and opening the splash screen (using the CSplashWnd class). A Splash Screen is a term known to persons skilled in the art and is the first image that pops up when a user starts a software program in Windows. This screen frequently includes the name of the application, the version, serial number and additional title related information.

EappFrame

In one embodiment, the main frame may handle the window commands (including opening and closing of windows), and the closing of the application.

General Features

Toolbars

In one embodiment, there are three toolbars in the authoring tool 1001 which are non-standard MFC toolbars, each of which is an EToolBox 44,003 derived classes. The three are the Workshop Tools, the Hotspot Tools 9001 and the Workshop Bar which is illustrated in Figure 8b and may be attached to the workshop window 8001.

In one embodiment, EToolBox 44,003 is a CDialogBar derived class (The CDialogBar class is a standard MFC class known to persons skilled in the art of Windows Programming) . In one embodiment, the EToolBox object 44,003 may handle an array of buttons, of type EToggleButton 44,005, and a background image. In one embodiment, the EToolBox 44,003 object may locate the buttons, and may handle the painting of the buttons and the background.

In one embodiment, EToggleButton 44,005 may be a CButton derived class (The CButton class is a standard MFC class known to persons skilled in the art of Windows Programming), which may have three states: normal, disabled and checked. In one embodiment, each state corresponds to a different background image which may be displayed when the state is operative. In one embodiment, EToggleButton 44,005 also may handle mouse moving and actuating.

1. The Workshop Tools bar may be of type EToolBox 44,003.

2. The Hotspot Tools 9001 bar may be of type EObjectsBar 44,006 (derived from EToolBox 44,003). In one embodiment, this class adds some additional methods to handle the hotspots 8003 combo-box (a combo-box is a Windows (TM) API standard control that is known to persons skilled in the art of Windows (TM) Programming).

3. The Workshop Bar illustrated in figure 8b, is of type EWorkshopBar 44,008 (derived from EToolBox 44,003). In one embodiment, this class adds some additional methods to handle the workshop slider 6007. In one embodiment, it also uses ESelectionButton 44,009 (derived from CButton) for the "Go to-In" and "Go to-Out" buttons which, In one embodiment, instruct the Workshop Window 8001 to display the IN and OUT frames of the currently displayed video. In one embodiment, the EWorkshopBar object uses the ESlider 44,007 class for the frames slider 6007. ESlider is a CSliderCtrl derived class, which, in one embodiment, adds some multi-frame selection functionality (The CSliderCtrl class is a standard MFC class known to persons skilled in the art of Windows (TM) Programming).

Run-Time Module

The run-time 1101 modules are responsible for presenting the content of an OBV file. In one embodiment, the run-time 1101 environment consists of the following seven modules:

- Mciva.dll which is the generic run time library 1113
- V-Active.ocx 1105 which is an ActiveX control to, for example, embed a hypervideo
  10,007 player in an HTML file in a browser.
- Npva.dll
- Npva.class
- va32.x32 1103 which is the Macromedia Director (Macromedia, Inc., San Francisco, CA) plug-in.
- VActiveRenderer.ax which is a standard DirectShow (Microsoft Corp., Redmond, WA) graphics/video renderer.
- VAVisual.ax which is a standard DirectShow (Microsoft Corp., Redmond, WA) graph filter.

MCIVA.DLL INTRODUCTION

In one embodiment, the mciva.dll 1113 is the core of all run-time 1101 environments and applications 1102. It may control the flow of the hypervideo 10,007 project 48001. In one embodiment, the mciva.dll 1113 maintains the communication with the viewer and implements the multimedia operations and performs all of the media manipulations 1125 (e.g. marking 1155). In one embodiment, the mciva.dll 1113 is an applications extension which may have only one exportable function called DriverProc. One embodiment of a DriverProc function will now be described.

In one embodiment, the prototype of the DriverProc function may be:

```
extern "C" LONG FAR PASCAL DriverProc(DWORD dwDriverID,
                HDRVR hDriver, WORD wMessage, LONG lParam1,
                LONG lParam2).
```

In one embodiment, the first two parameters are ignored. The parameter wMessage determines the operation to be performed by the DriverProc function.

In one embodiment, The mciva.dll 1113 may support driver operations and MCI 1109 operations.

In one embodiment, the supported driver operations are: load, free, open and close which will now be described:

Load performs any one-time driver initialization (wMessage gets DRV_LOAD).

Free performs any one-time shutdown tasks (wMessage gets DRV_FREE).

Open performs initialization, done once each time the driver is opened (wMessage gets DRV_OPEN).

Close performs any cleanup necessary each time the device is closed (wMessage gets DRV_CLOSE).

In one embodiment, the MCI 1109 operations supported are: Open, close, play, stop, pause, window, status, signal object, internet, put, enable, disable, media, visualize, sample and set flag. These operations will now be described:

MCI OPEN DRIVER: wMessage - MCI_OPEN_DRIVER

lParam1 - flags passed with the MCI_OPEN command

lParam2 - a pointer to a MCI_DGV_OPEN_PARMS

In one embodiment, this operation may be called whenever a new project 48,001 is to be loaded. The steps for opening a new project 48,001 may include: initialization of global variables, right button menu construction (play/stop), preview-window construction, hypervideo 10,007 project file initialization and project 48,001 construction. In one embodiment, when constructing the project 48,001, the hypervideo 10,007 project file version

may be checked. In one embodiment, old versions may be supported, but presenting an hypervideo 10,007 project of a newer version then that of the object database 1127 may not be supported. In one embodiment, after constructing the project 48,001 , the first level 48,015 of the project 48,001 may opened and the run-time 1101 level tree illustrated in figure 13 may be created wherein the root 13,006 may be the first target level 48,015 of the project 48,001. In another embodiment, if there is no first level or the opening of the tree fails MCI close driver is activated and the operation fails.

MCI CLOSE DRIVER: wMessage - MCI_CLOSE_DRIVER
lParam1 - flags passed with the MCI_CLOSE command
lParam2 - a pointer to a MCI_DGV_OPEN_PARMS
In one embodiment, if the project 48,001 is not already opened, then no operation may is performed. In another embodiment, a termination procedure may performed. The termination procedure may include preview-window destruction, levels tree 13,004 destruction and project 48,001 destruction.

MCI PLAY: wMessage - MCI_PLAY
lParam1 - flags passed with the MCI_PLAY command

lParam2 - a pointer to a MCI_DGV_PLAY_PARMS

In one embodiment, if the project 48,001 is not already opened, or the driver 1109 is already playing, then no operation is performed. In one embodiment, if the driver 1109 is in the >pause= state, then the levels tree 13,004 may be searched and the paused levels 48,015 may be played. Otherwise, the first level 13,006, (the root of the levels tree 13,004) may be played. In one embodiment, if MCI_NOTIFY is passed in lParam1, then a new notification window may be created. A notification window is an object known to persons skilled in the art of Windows programming. Otherwise, it may use the notification window supplied.

MCI stop: wMessage - MCI_STOP

lParam1 - flags passed with the MCI_STOP command

lParam2 B ignored.

In one embodiment, if the project 48,001 is not already opened, or the driver 1109 is not playing, or the driver 1109 is in pause state, then no operation may be performed. Otherwise, the driver 1109 may be switched to a stop state. If lParam1 equals 1 then a message is posted to the background window. The actual driver 1109 stop may be activated from the message handler (by calling the driver=s MCI_STOP command but with lParam1 equals 0). This mechanism allows

processing of waiting messages before driver 1109 is stopped.

MCI PAUSE: wMessage - MCI_PAUSE
lParam1 - flags passed with the MCI_PAUSE command.
lParam2 B ignored.
In one embodiment, if the project 48,001 is not already opened, or the driver 1109 is not playing, then no operation may be performed. Otherwise, the driver 1109 may be switched to a pause state.

MCI CLOSE: wMessage - MCI_CLOSE
lParam1 - flags passed with the MCI_CLOSE command.
lParam2 B ignored.
In one embodiment, if a project 48,001 is currently opened, it may be closed. Closing a project 48,001 may include the following steps: Switching the run-time tree 13,004 to a stop state, closing the run-time tree 13,004, destroying the run-time tree 13,004 object, destroying the project 48,001 object, destroying the background window and sending a message to the notification window. The message queue emptying mechanism, as in the MCI_STOP command, is implemented.

MCI WINDOW: wMessage - MCI_WINDOW
lParam1B the window attribute to be changed.
lParam2 B a pointer to a MCI_DGV_WINDOW_PARMS.

In one embodiment, this operation may be used to set the background window attributes such as: Windows visibility and Windows text.

MCI STATUS: wMessage - MCI_STATUS
lParam1B the driver attribute to be retrieved.
lParam2 B a pointer to a MCI_DGV_STATUS_PARMS.
In one embodiment, this operation may be used to retrieve driver attributes.

MCI PUT: wMessage - MCI_PUT
lParam1B MCI put operation flags.
lParam2 B a pointer to a MCI_DGV_PUT_PARMS.
In one embodiment, this operation may be used for repositioning background window.

MCI INTERNET: wMessage - MCI_INTERNET
lParam1B The value of the internet flag.
lParam2 B ignored.
In one embodiment, this operation may be used for setting the value of the internet flag.

MCI ENABLE: wMessage - MCI_ENABLE
lParam1B Holds the media object's name.
lParam2 B Holds the media name.
In one embodiment, this operation may be used to browse the object database 1127 illustrated in figure 48 and enable the appropriate media objects 48,026. In one embodiment, a hotspot 8003, or media object that is enabled may be marked with a

marking 1155 filter in the run-time module 1101 and may be linked to a target 7003 at run-time upon actuating the mouse button (e.g. the mouse button is clicked) on the hotspot 8003.

MCI DISABLE: wMessage - MCI_DISABLE
lParam1B Holds the object=s name.
lParam2 B Holds the media name.
In one embodiment, this operation may be used to browse the object database 1127 illustrated in figure 48 and disable the appropriate media objects 48,026. Disabling a media object, or hotspot 8003, may prevent the run-time, for example, from marking 1155 the hotspot 8003 at run-time with a marking 1155 filter and may prevent the run-time module 1101 to link the hotspot 8003 to a target 7003 upon actuating a mouse button on the hotspot 8003.

MCI MEDIA: wMessage - MCI_MEDIA
lParam1B Holds the object's name.
lParam2 B Holds the media name.
In one embodiment, this operation may return Boolean 'TRUE' if the specified media is currently playing.

MCI SAMPLE: wMessage - MCI_SAMPLE
lParam1B may hold a pointer to the video object.
lParam2 B may hold a pointer to a SAMPLE_DATA.
In one embodiment, this operation may be called by the renderer (described above) or the marking 1155

filter (described above) upon rendering a video frame. In another embodiment, this operation may be used for time calculations and marking 1155.

MCI DESTROY NOTIFY:

wMessage - MCI_DESTROY_NOTIFY
lParam1 B ignored.
lParam2 B ignored.
In one embodiment, this operation may be used for destroying the notification window.

MCI SET FLAG:

wMessage - MCI_SET_FLAG
lParam1 B the flag to update.
lParam2 B the new value of the flag.
In one embodiment, the following flags may be updated using this MCI command:

1. 'use DirectShow' (when TRUE BDirectShow is used, when FALSE B MCI is used)
2. 'allow visualization' and 'run-time environment' (when TRUE B the run-time environment is not the program authoring tool 1001).

THE RUN-TIME TREE 13,004

An exemplary design of the run-time tree will now be described: In one embodiment, while previewing an hypervideo 10,007 project 48,001 file, the run-time module 1101 may maintain a levels tree 13,004. In one embodiment, the nodes of the run-time tree 13,004 may be the currently active target levels 48,015 (i.e. Target Levels 48,015 that are currently opened). In another embodiment, each node represents a target level 48,015 and may hold an array of objects

13,002. Each object 13,002 may control the playing of a media file. In one embodiment, the type of the media file may be, for example, one of the following: Video 1011, Audio 1015, Bitmap 1013, Text 1017, Executable (.exe file) 1021, and URL 1019.

In one embodiment, each operation performed on a level 48,015 (e.g. stop, play, open etc.) may also be performed on each object 13,002 in that objects array. In one embodiment, the root of the tree may be a dummy node 13,006 that may have no objects 13,002 in its objects array. In one embodiment, when a new project 48,00 is opened (during the execution of MCI_OPEN command) the first real node 13,002 may be constructed from the default target level 48,015 of the project 48,001 (known as the Base Target and is constructed as a son of the dummy node). In another embodiment, each level (node) may support the following operations: Open, Close, Play, Stop, DriverPlay, DriverStop, Pause, Resume. Additional operations may be:

Window 13,002	In one embodiment, this operation may set the node's window
ParentLevel	In one embodiment, this operation may return a pointer to the node's 13,002 parent
TargetLevel	In one embodiment, this operation may return a pointer to the node's 13,002 EtargetLevel 48,015 data member.
NextTargetLevel	In one embodiment, this operation may return a pointer to the node's 13,002 next target level 48,015
MainTargetIndex	
Previous	In one embodiment, this operation may return a code indicating what to do with the previous active node 13,002 (e.g. keep playing, pause, stop, etc.)
IsMainTarget	In one embodiment, this operator may check if an object 13,002 is the main target 48,009 (i.e. the leader).
DeleteSubTree	In one embodiment, this operation may delete the sub-tree under it

PlaySubTree	In one embodiment, this operation may play the sub-tree under it
StopSubTree	In one embodiment, this operation may stop the sub-tree under it
DriverPlaySubTree	In one embodiment, this operation may play the sub-tree under it as a result of a driver command
DriverStopSubTree	In one embodiment, this operation may stop the sub-tree under it as a result of a driver command
CloseSubTree	In one embodiment, this operation may close the sub-tree under it

In one embodiment, when a level 13,002 is currently playing the following exemplary scenarios can occur:

1. One target 48,009 may have finished normal execution (e.g. a video that finished). In one embodiment, when a media finishes playing, it may send a message to the notification window. The message handler may query the object 13,002 what to do next. An example of options for operations to be executed when an object 13,002 finishes playing will now be described. In one embodiment the options may be:

Exit	In one embodiment, the playback of the project 48,001 may be stopped and closed using the MCI_CLOSE_DRIVER command.
Loop	In one embodiment, this may require no changes because the default behavior of an object may be set to loop (repeat playback).
Continue	In one embodiment, the active level 48,015 may be queried for the next target level 48,015. In another embodiment, if a next target level 48,015 exists, then a new node 13,002 may be constructed from the new target level 48,015 as a "brother" to the current active level 48,015. In one

embodiment, the current active level 48,015 may be stopped. The new level 48,015 may be opened and played. In another embodiment, the previous level 48,015 may be closed and removed from the tree 13,004. In one embodiment, if the playing of the new level 48,015 fails, then the parent is resumed.

Back — In one embodiment, the current active level 48,015 is stopped, closed and removed from the tree 13,014. The parent level 48,015 is resumed.

2. In one embodiment, one of the defined hotspots 8003 may have been actuated, for example, by the user clicking the mouse button on the hotspot 8003. In one embodiment, hotspots 8003 can be clicked using the left button of the mouse. In one embodiment, a message is sent to the messages handler for the window (messages, message handlers and windows are known to persons skilled in the art of Windows (TM)programming). The handler of the window that received the "left button down" message may identify the object 13,002 on which the clicked hotspot 8003 is defined. In one embodiment, when clicking on a hotspot 8003, the configuration of the cursor changes. The object 13,002 retrieves the new cursor configuration and next target level 48,015 by contacting the object database 1127 for the project 48,001. In one embodiment, if the user clicked on an unmarked area, then the object may return a null as the new cursor's handle and the new target level 48,015. In this embodiment, the new target level 48,015 and the new cursor 48,027 are ignored and no changes occur. In one embodiment, if a new cursor 48,027 exists, then it may be presented. In one embodiment, if a target level 48,015 exists then its type is checked. If it is an exit target level 48,015 then the project 48,001 may be stopped and closed. If it is a back target level 48,015, then the previous level's 48,015 node 13,002 may be closed and deleted from the tree 13,004, and its parent may be

resumed. Otherwise, a new node 13,014 may be constructed from the new target level 48,015, as a son of the current level's node 13,002. The new level 48,015 is opened and played. The new level 48,015 may return a code indicating what to do with the previous level 48,015. The previous level 48,015 can be paused, stopped and closed or continue playing.

3. The entire project 48,001 was paused using the MCI_PAUSE command. In one embodiment, the method DriverPause may be activated on each node 13,002 in the run-time tree 13,004. In one embodiment, every active media is paused. Later, when the project 48,001 is resumed, each node 13,002 in the tree 13,004 is resumed.

One embodiment of the run-time tree class diagram is depicted in figure 45. This diagram shows the existence of classes and their relationships.

Figure 45 represents the run-time tree class diagram in general. The different items of diagram 45 will now be described in the following table

Item #	Class Name	Attributes	Operations
45,009	EVAObject	m_pTarget:Etarget m_pLevel	
45,001	ENode	m_oChildren:CobjList m_pParent:ENode	void DeleteSubTree() void TakeOfChild() void ConnectChildrenToParent() void AddChild() void SetParent()
45,008	EVALevel	m_pTargetLevel:EtargetLevel m_oObjects:EVAObjectArray m_oMousePosition:Cpoint m_oRefRect:Crect m_lOffsetFrames:long	
45,007	EVABitmapObject	m_pDiB:Edib m_pOParms:EbitmapOParms m_oBitmapStatus:EbitmapStatus m_lStartTick:long	
45,006	EVAExecObject	m_pOParms:EexecOParms m_oExecStatus:EexecStatus m_pProcessInfo:PROCESS_INFORMATION	
45,005	EVASoundObject	m_pWave:Cwave m_pOParms:EsoundOParms m_oSoundStatus:ESoundStatus	

45,004	EVATextObject	m_pText:char m_nTextLength:int m_pOParms:EtextOParms m_oTextStatus:EtextStatus m_IStartTick:long
45,003	EVAURLObject	m_pOParms:EURLOParms m_oURLStatus:EURLStatus m_oNavigate:NAVIGATEREC m_bTargetAllocated:BOOL
45,002	EVAVideoObject	m_pMovie:CmyMovie m_pMediaNode:EmediaNode m_pOParms:EvideoOParms m_oVideoStatus:EvideoStatus m_hWaitHandle:HANDLE m_sFileName:Cstring m_lMediaPosition:long m_lIOrgFormatMediaPos:LONG LONG m_pMarking 1155:EVAMarking 1155

The arrow line between item 45,001 and item 45,008 denotes that 45,008 is derived from 45,001.

Items 45,002, 45,003, 45,004, 45,005, 45,006 and 45,007 are derived from 45,009.

THE VIDEO OBJECT

In one embodiment, the video object, implemented by the EVAVideoObject class 45,002 may control the presentation of a video file. In one embodiment, the EVAVideoObject may hold a pointer to an object of class CVAMovie, which is an abstract class that defines the interface to the different objects that actually performs the video operations. The run-time module 1001 may, for example, support two multimedia environments: MCI and DirectShow (previously called ActiveMovie).

MCI is one multimedia environment. The run-time module 1001 communicates with the video type MCI drivers by a single API function called mciSendCommand and is known to persons skilled in the art of Windows programming.

DirectShow is another multimedia environment that may control and processes streams of time-stamped multimedia data by using modular components, called filters, connected in a configuration, called a filter graph. This architecture is known to persons skilled in art of Windows programming.

The run-time module 1001 may, for example, support the following formats of video files: AVI, MPEG-1 and Quick-Time.

In one embodiment, the following classes control movies in the DirectShow environment: CVAMovieAvi, CVAMovieMpeg and CVAMovieQt each corresponding to a particular video format.

In one embodiment, the CVAMovie may define the following operations:

FileOpen	In one embodiment, this operation may open the video file
MovieOpen	In one embodiment, this operation may open the movie in a given window. When opening a video file in DirectShow the EVAVideoObject 45,002 first attempts to load the VActiveRenderer filter discussed above. In one embodiment, upon success, the EVAVideoObject 45,002 may add the renderer filter to the DirectShow filter graph. In one embodiment, upon failure, the EVAVideoObject 45,002 may attempt to load the marking 1155 filter. In another embodiment, after adding the renderer, or the marking 1155 filter, to the DirectShow graph, the DirectShow graph may attempt to render the movie file using the new filters added. This is standard operation of the DirectShow Software Program. If the DirectShow

	graph cannot render the movie file neither with the renderer filter nor with the marking 1155 filter, then it may render the movie file anyway using its default filters. In this case marking 1155 may not be performed, and the movie is presented using the default supplied DirectShow video renderer (which may have inferior performance).
MovieSetDestOptions	In one embodiment, this operation may put the video at a certain position.
MoviePlay	In one embodiment, this operation may play the video entirely or play only a specified segment (i.e. range of frames).
MovieStop	In one embodiment, this operation may stops the presentation of a movie.
MoviePause	In one embodiment, this operation may pause the presentation of a movie. This means that the movie may be stopped and the number of the last frame played before the movie is stopped may be saved.
MovieResume	In one embodiment, this operation may start playing the movie from the position it was previously paused, for example, by the MoviePause operation described above.
MovieClose	In one embodiment, this operation may close the currently opened movie file. This may include closing of the driver, and deletion of objects.

MovieGetPosition	In one embodiment, this operation may get the current playing frame number.
SetSignal	In one embodiment, this operation may sets the video to signal on frames. This means that the video playback mechanism will now signal back to the object that a frame is being played.
FrameToTimeFormat	In one embodiment, this operation may convert frame number to playing time, for example, in a specified time unit (e.g. milliseconds).
TimeFormatToFrame	In one embodiment, this operation may converts playing time to frame number.

One embodiment of the EVAVideoObject 45,002 class diagram is depicted in Figure 46. The different features of Figure 46 and the relationships between them will now be described in the following table:

Item #	Class Name	Attributes	Operations
46,001	CVAMovie		FileOpen() MovieOpen() MovieSetDestOptions() MoviePlay() MovieStop() MoviePause() MovieClose() FileClose() MovieSeek() MovieResume() MovieGetPosition() SetSignal() MovieSignal() MovieUpdate() MovieWindow() FrameToTimeFormat() TimeFormatToFrame()

46,009	EVideoObject	m_pMovie:CVAMovie m_pMediaNode:EmediaNode m_pOParms:EvideoOParms m_oVideoStatus:Evideo Status m_hWaitHandel:HANDLE m_sFileName:CString
46,008	CVAMovieMciAvi	m_wDeviceID:WORD m_hWnd:HWND m_sFileName:CString
46,007	CVAMovieMciMpeg	m_wDeviceID:WORD m_hWnd:HWND m_sFileName:Cstring m_IFrameRate:long
46,005	CVAMovieMciQtw	m_wDeviceID:WORD m_hWnd:HWND m_sFileName:Cstring m_IFrameRate:long
46,002	CVAMovieAvi	m_oMilliSecFrameLength:REFTI ME m_IRoundError:long, m_IIStartTime:LONGLONG m_bUseVisualFilter:BOOL m_bUseVActiveRenderer:BOOL m_oMediaType:MEDIA_TYPES m_pVActiveRenderer:IFilterm_h Wnd:HWND m_sFileName:Cstring
46,004	CVAMovieMpeg	m_IDuration:LONGLONG m_INumFrames:LONGLONG
46,003	CVAMovieQt	M_IUnitTimeLength:long

The arrow line between item 46,009 and item 46,001 denotes the relationship between these items. The arrow line between item 46,008 and item 46,001 indicates that 46,008 is a class derived from 46,001. The arrow line between item 46,007 and item 46,001 indicates that 46,007 is a class derived from 46,001. The arrow line between item 46,005 and item 46,001 indicates that 46,005 is a class derived from 46,001. The arrow line between item 46,002 and item 46,001 indicates that 46,002 is a class derived from 46,001. The arrow line between item 46,004 and item 46,002 indicates that 46,004 is a class derived from 46,002. The arrow line between item 46,003 and item 46,002 indicates that 46,003 is a class derived from 46,002.

THE SOUND OBJECT

In one embodiment, the sound object , implemented by the EVASoundObject class 45,005 may control the playing of wave files. Wave files are sound file

format that is known to persons skilled in the art. The EVASoundObject 45,005 may further direct a pointer to an object of class CWave, which is an abstract class that defines the interface to the different objects that actually perform the playing of the sound. Sounds can be played using MCI or DirectShow environments as discussed above in the case of video playback. The sound object EVASoundObject 45,005 may support the following operations:

Open	In one embodiment, this operation may open a sound file and create a window to receive the message sent when the sound finishes.
Close	In one embodiment, this operation may close the sound file, including, deletion of objects.
Play/Stop	In one embodiment, this operation may play or stop playback of a sound file.
Pause/Resume	In one embodiment, this operation may pause or resume the playing of a sound file.
DriverPlay/DriverStop	
Window	In one embodiment, this operation may set the window for the sound object.
MediaName	In one embodiment, this operation may return the media name of the corresponding sound file.
GetMediaPosition	In one embodiment, this operation may get the current position in the wave file (e.g. in time units).
IsPlaying	In one embodiment, this operation may return Boolean 'TRUE' if the wave file is currently being played.
Previous	In one embodiment, this operation may return a code indicating what to do with the sound object EVASoundObject 45,005 when another object is scheduled for playback after it.
Notify	In one embodiment, this operation may return a code indicating what to do when a sound object

EVASoundObject 45,002 may have finished playing.

One embodiment of the sound object class diagram is depicted in figure 47. The different items of diagram 47 and the relationships between them will now be described in the following table:

Item #	Class Name	Attributes	Operations
47,001	CWave		Load() Play() Stop() Close()
47,004	EVASoundObject	m_pWave:Cwave m_pOParms:EsoundOParms m_oSoundStatus:ESoundStatus	
47,003	CWaveVA	m_wDeviceID:WORD	
47,002	CWaveAM	m_pWinThread:CwinThread m_pGraph:IgraphBuilder m_hWnd:HWND m_pLock:CCritSec	

The arrow line between item 47,004 and item 47,003 denotes the relationship between these items. The arrow line between item 47,003 and item 47,001 indicates that 47,003 is a class derived from 47,001. The arrow line between item 47,002 and item 47,001 indicates that 47,002 is a class derived from 47,001.

THE TEXT OBJECT

In one embodiment, the text object, implemented by the class EVATextObject 45,004 may control the presentation of a text file. In one embodiment, when playing a text object 45,004 the content of the text file may be painted, for example, on a window that was created when the text object 45,004 was opened. In one embodiment, the text may disappear after a predefined amount of time. The text object 45,004 may support the following operations:

Open	In one embodiment, this operation may open the text file, download it from the internet if needed, create a window and set its position.
Play	In one embodiment, this operation may switch the window into a 'show' state which is a Windows API state for a window. This terminology is known to persons skilled in the art. In one embodiment, this operation may also draw the text on the window's device context (Abbreviated 'DC' and known to persons skilled in the art. In one embodiment, this operation may also activate a standard Windows timer to send a message after a predefined amount of time.
Stop	In one embodiment, this operation may 'kills' the timer procedure.
DriverPlay	
DriverStop	
Pause	In one embodiment, this operation may 'kill' the timer procedure, calculate and save the time remaining for the text to be presented.
Resume	In one embodiment, this operation may set a timer according to the time calculated in the last pause operation.
Close	In one embodiment, this operation may destroy the text window and delete remaining objects.
Update	In one embodiment, this operation may draw the text on the Window's DC discussed above.
Window	In one embodiment, this operation may sets the text object's 45,004 window.
GetTargets	In one embodiment, this operation may return a pointer to an EtargetLevel object 48,015 according to the position of the mouse click and the frame number.
Timer	In one embodiment, this operation may return a code indicating what to do when the timer (activated during the play operation) sends its message.

THE BITMAP OBJECT

In one embodiment, the bitmap object, implemented by the class EVABitmapObject 45,007 may control the presentation of a bitmap file in the hypervideo 10,007. In one embodiment, when opening a bitmap object 45,007 a window may be created. In one embodiment, the size and position of the window may be set according to the desired size and position of the bitmap. In one embodiment, when playing a bitmap object 45,007 the bitmap may be painted on the new window. In addition, according to one embodiment, a timer is created with a time-out value read from the project's 48,001 object database 1127. When the time-out occurs, a message is sent to the object's 45,007 window. In one embodiment, the message handler may process the time-out event in a manner similar to the MM_MCINOTIFY (sent after a video or a sound file ends. MM_MCINOTIFY is a Windows API message known to persons skilled in the art). In one embodiment, the bitmap object 45,007 may hold a pointer to an object of class EDib which is responsible for the actual displaying of the bitmap (EDib is a service class for opening and decompressing, for example, .bmp, .gif and .jpg). In one embodiment, the bitmap object 45,007 may support the following operations:

Open	In one embodiment, this operation may creates the window, set the size and position according to the project's 48,001 object database 1127. In one embodiment, this operation may also determine if the bitmap file is a local file or if it requires downloading from the Internet. In one embodiment, this operation may also download the bitmap file if necessary, create an object of class EDib, which opens and loads the bitmap file.
Play	In one embodiment, this operation may paint the bitmap, using the EDib object, on the windows device context (DC. Discussed above). In one embodiment, this operation my also create a timer and set its time-out value according

	to the project's 48,001 object database 1127. In one embodiment, this operation may also notify the parent application that the media may have begun playing.
Stop	In one embodiment, this operation may 'kill' the timer procedure and notify the parent application.
DriverPlay	
DriverStop	
Pause	In one embodiment, this operation may perform the same operation as the Stop operation. In addition, according to one embodiment, this operation may store the time remaining for the time-out to occur.
Resume	In one embodiment, this operation may play the bitmap object 45,007. In one embodiment, this operation may also set the timer according to the time calculated in the last pause command.
Close	In one embodiment, this operation may deletes the EDib object and destroy the window.
Update	In one embodiment, this operation may repaints the bitmap.
IsPlaying	In one embodiment, this operation may return Boolean 'TRUE' if the bitmap object 45,007 is currently playing.
GetTargets	In one embodiment, this operation may return the ETargetLevel 48,015 to be opened when, for example, the mouse button is clicked on a hotspot 8003 on the Bitmap Object 45,007.
MediaName	In one embodiment, this operation may returns the media name of the Bitmap Media object 45,007 currently being handled..
Timer	In one embodiment, this operation may return a code indicating what to do upon successful notification (for example, exit, freeze, keep playing). This function is called upon timer notification.

GetCursor	In one embodiment, this operation may return the handle of the cursor to be presented. It may return the move cursor or the click cursor.
OnSize	In one embodiment, this operation may handle dimensions changing in the preview window.

THE URL OBJECT

In one embodiment, the run-time module 1101 may support web navigation by activating Targets 7003 that are URL Targets 1019. In one embodiment, the viewer can navigate the web by clicking the mouse on hotspots 8003 on video, on bitmap or other media. In one embodiment, each time a URL object 45,003 is played, a message is sent to the parent application via the notification window. In one embodiment, the parent application may be an Internet Browser. In one embodiment, the responsibility for the actual web navigation may be on the parent application. In one embodiment, the URL object 45,003 may pass a pointer to a NAVIGATEREC object. The NAVIGATEREC structure may contain the URL, the location in the URL and the HTML frame name in case this URL is a multi-frame. The URL object 45,003 may support the following operations:

Open	In one embodiment, this operation may create the NAVIGATEREC object and initialize its fields as read from the project's 45,001 object database 1127.
Play	In one embodiment, this operation may send the NS_NAVIGATE message to parent application, for example, Internet Browser with a pointer to the NAVIGATEREC object as a parameter.
Stop	In one embodiment, this operation may update the object's 45,003 status flags.
DriverPlay	
DriverStop	

THE EXECUTABLE OBJECT

In one embodiment, the run-time module 1101 may support executable files as media targets 7003. Executable files can be activated, for example, by operations such as a hotspot 8003 being clicked on or a Target Level 48,015 finishing execution. In one embodiment, the executable object 45,006 may use the CreateProcess standard Windows API function to run the executables. In one embodiment, the executable object 45,006 may use a timer to periodically check the status of the new process. In one embodiment, periodic checking occurs every 100 milliseconds. In another embodiment, a worker thread is used rather than a timer. In this embodiment, the thread loops repeatedly and with each iteration, the status of the new process is checked. The executable object may support the following operations:

Open	In one embodiment, this operation may create a window for receiving the timer notifications. In this embodiment the operation may also read the executable attributes from the project 48,001 object database 1127.
Play	In one embodiment, this operation may create a process in which to run the executable. This operation may also create a timer to notify, for example, each 100 milliseconds.
Stop	In one embodiment, this operation may update the status parameters of the object 45,006.
Close	In one embodiment, this operation may 'kill' the timer. This operation may also check the status of the process, and if still active, terminate it. In one embodiment, this operation may also destroy the notification window.
Timer	In one embodiment, this operation may handle the timer notifications. In this embodiment the operation may also check the status of the new process.

HOTSPOT MARKING

In one embodiment, the run-time module 1101 may support object marking 1155 when using DirectShow. In this embodiment the marking 1155 may be performed on each video frame just before it is rendered. In one embodiment, the video frames are passed to the run-time module 1101 by the renderer or the marking 1155 transform filter. According to this embodiment, if the recent filters are not included in the filter graph, then marking 1155 cannot be performed. Further, according to this embodiment, if a default DirectShow renderer is used then the transform filter is automatically added to the graph.

In another embodiment, marking 1155 may also be performed using the program video renderer. According to this embodiment, if the V-Active video renderer is loaded, then it may not be necessary to load the transform filter. Marking 1155 may be performed in the run-time module on video samples from the DirectShow filter (either video renderer or transform filter) just before rendering. In one embodiment, marking 1155 may be performed on samples, for example, of the following color depth: 16-bit, 24-bit and 32-bit.

In one embodiment, marking 1155 may be performed by an object of type EVAMarking. According to this embodiment, the EVAMarking object may hold a pointer to the video object 46,009 that created it. The pointer may be used to query the project's 48,001 object database 1127 whether a hotspot 8003 exists in a specific point on a specific frame, and whether this hotspot 8003 should be marked 1155. The following classes inherit from EVAMarking : EVA32BitMarking, EVA24BitMarking, EVA555Marking and EVA565Marking. These classes perform the different marking 1155 effects in the supported frame types.

Project File

Active Object database 1127

1. Introduction

In one embodiment, the object database 1127 may be a software package which maintains all the data of a project 48,001 and enables the authoring tool 1001 and run-time 1101 to fetch, modify and serialize this data. In one embodiment, a

project 48,001 is actually a tree of Targets 48,009. Each Target 48,009 (link) is attached to a Media (for example, video 48,002, picture 48,003, sound 48,004, or text) and its Operating Parameters 48,023-48,025 (for example, where and how the media should be displayed, and what happens when the media starts to perform and when it ends). However, according to this embodiment, a particular media can be attached to more than one target, so the tree may be infinite. In one embodiment, a targets-group may contain the targets 48,009 that should be performed and the order and manner of their performance. According to this embodiment, moving from one targets-group to another is done using hotspots 8003, 48,030. Some media types may contain hotspots 8003. For example, in a picture, the hotspot 8003 may be a geometric shape inside the boundaries of the picture. In one embodiment, in a video, the hotspot 8003 may also be a geometric shape, but it can change its size and shape throughout the frame of the video. In this embodiment, the hotspot 8003 may exist only in a part of the video (i.e. only in some of the video frames). In one embodiment, each hotspot object 48,030 in the object database 1127 points to a certain targets-group. This targets-group is performed when the hotspot 8003 is clicked using the mouse buttons.

In one embodiment, another object in the project 48,001 object database 1127 is a cursor object 48,027 which may be representing a standard Windows-cursor. According to this embodiment, each hotspot 8003 may have two related cursors - a move cursor and a click cursor. The move cursor is the cursor that is set when the user 1149 moves the mouse over the hotspot 8003. The click cursor is set when the user 1149 clicks on the hotspot 8003. In one embodiment, in order to avoid data duplication, the project 48,001 stores one array of cursors 48,027 and all the hotspot objects 48,030 hold pointers to indices of this array.

An exemplary class structure for the project 48,001 object database 1127 1127 will now be described:

2 Classes

The following table describes the classes that are defined by the hypervideo 10,007 object database 1127 and their functionality. Figure number 48 shows the

class hierarchy and the relationships between the classes in the object database 1127.

Class Name	#	Functionality in this embodiment
EProject	48,001	Contains all the data of the project. Contains, for example, file name, paths, cursors array, and other general information
EMedia	48,002-48,005	Contains data of a certain media. Contains, for example, the name of the media file. The actual media is read from a file and in one embodiment, is not maintained by the object database 1127. EMedia may have a number of derived classes, wherein each derived class may contain an instance of its appropriate operating parameters. Examples include:
EBitmapMedia	48,003	Contains data of a bitmap type media, including, for example, the following file types: BMP, DIB, GIF, JPG
EExecMedia		Contains data of an executable file type of media element, including, for example, the EXE file types
ESoundMedia	48,004	Contains data of a sound type media, including, for example, the following file types: WAV, AIF, AU
ETextMedia		Contains data of a text type media, including, for example, the following file types: TXT, RTF
EURLMedia		Contains data of a URL type media, including, for example, the HTM and HTML file types as well as a string containing a URL address on the World Wide Web.
EVideoMedia	48,002	Contains data of a Video type media, including, for example, the following file types: MPG, AVI, MOV
EMediaOParam	48,023-48,026	Contains data pertaining to Operating parameters of a media object. EMediaOParam may have a number of derived classes, wherein each derived class may contain an instance of its appropriate operating parameters. Examples include:
EBitmapOparams	48,024	Contains data pertaining to operating parameters of an EBitmapMedia object.
EExecOParams		Contains data pertaining to operating parameters of an EExecMedia object.
ESoundOParams	48,023	Contains data pertaining to operating parameters of an ESoundMedia object.

ETextOParams		Contains data pertaining to operating parameters of an ETextMedia object.
EURLOParams		Contains data pertaining to operating parameters of an EURLMedia object.
EVideoOParams	48,025	Contains data pertaining to operating parameters of an EVideoMedia object.
EClip	48,022	Contains data about a clip. A clip is a class that resides as a member of a, for example, video or bitmap media and contains the hypervideo 10,007 information for the media objects.
EClipObject	48,026	Contains static information of a hotspot for a certain media file. This class holds information that is repeated for all instances of hotspots 8003 throughout the frames of a video.
EClipFrame	48,006	Represents a frame in a video and contains all hypervideo information for that frame in the video. This object is a member of in an array of its type inside an EClip object.
EFrameObject	48,021	Representation of an instance of a hotspot in one frame of the video. This class includes a pointer to the static hotspot information (i.e. a pointer to an EClipObject object) as well as a geometric region for the particular hotspot 8003 in the particular frame of the video.
EGeometry		Representation of a geometric shape which describes the region of the hotspot in a particular frame or group of frames in the video. Derived classes of EGeometry, which contain the actual size and location of the shape, include:
EEllipse	48,020	Represents an elliptical region.
EPolygon		Represents an polygon region.
ERect	48,017	Represents an rectangular region.
ETriangle		Represents an triangular region.
ETarget	48,009	Contains data pertaining to a target 7003. This class may contain the media to be performed and its operating parameters. The default parameters are the operating parameters of the media object.
ETargetLevel	48,015	A target level object which is an array of all the targets which are performed simultaneously. A targets-group is constructed from a list of target-levels.
ECursor	48,027	Data of a cursor object including its file name and a handle to the windows cursor.
CPointArray		An array of x,y coordinates for use, for example, in defining a polygon region.

3 Project Serialization

In one embodiment, the hypervideo 10,007 Project 48,001 serialization may be done using the EProject object 48,001 *Load()* and *Store()* methods. According to this embodiment, each of these two functions may have two versions. The first set of functions may load (and store) the project from (or into) a memory area. The second does the same with a file. In one embodiment, these methods use the standard serialization procedure of the EProject class. After the project is stored in the memory area, the area is compressed using the function ECmprs0. Before loading the project from the area, the area must be decompressed using the function EDcmprs0.

Project Environment

In one embodiment, general information about the project serialization is stored in EProject 48,001 static members. According to this embodiment, this information may include the following static member variables which are described in the following table. For the sake of simplicity, the expression '*media files*' may also refer to cursor type files:

m_bRunTime In this embodiment, this member should be set before *Load()* is called. Set to FALSE if the project is loaded in an authoring tool 1001 environment (that is, the authoring tool 1001 reads the project, or the run-time of the authoring tool 1001=s preview reads the project). Otherwise, set to TRUE. If this member's value is set to FALSE, then the media relative paths will be added to the path of the OBV file (*m_sOBVPath*) and the custom paths will be ignored. That means that the original paths of the media files will be used (these are the paths from which the author may have read the media in the authoring tool 1001). Otherwise, the relative paths

will be added to the URL and CD paths, and custom paths will not be ignored. Thus the publish paths are used. Published paths are paths after the project may have been published.

m_sOBVPath According to this embodiment, this member should be set before *Load()* is called and also before *Store()* is called. This is the path of the OBV file. When the project 48,001 is stored, all media paths will be relative to this path. When the project 48,001 is loaded, the paths will be relative to this path unless *m_bRunTime* is TRUE.

m_sURLPath According to this embodiment, this member is stored within the OBV file. This is the base path for media files that come from the Internet. When the project is loaded, the paths of such media files will be relative to this path unless *m_bRunTime* is FALSE.

m_sCDPath According to this embodiment, this member is stored within the OBV file. This is the base path for local media files. When the project is loaded, the paths of such media files will be relative to this path unless *m_bRunTime* is FALSE.

m_cCDLetter According to this embodiment, this member should be set before *Load()* is called. This should be the actual drive letter of the CD-ROM in the current computer. If the OBV comes from the Internet, then the Drive-letter of the CD path will be changed to match. In this embodiment, it may be

assumed that the Internet OBV will use media from the CD.

m_bDownload According to this embodiment, this member may be stored within the OBV file. This member specifies the default source (local or Internet) of media files that do not have a specific instruction for determining where to get the file from.

m_wVersion According to this embodiment, this member is set by the EProject::*Load()* function to the version number of the OBV file currently loaded. When the project 48,001 is loaded, the serialization functions of all the object database 1127 classes check this static member to know the version of the OBV. Based on the version number, these functions know which data should be stored in the OBV.

m_bForceOBVPath According to this embodiment, this member is stored within the OBV file. If the value of this member is Boolean TRUE and the project 48,001 comes from the Internet, then the URL path will be set to the OBV path. Otherwise, the CD path will be set to the OBV path.

m_spCursorLoadProc According to this embodiment, this member should be set before *Load()* is called. This member may contain a pointer to a function of a form: *BOOL LoadCursorProc* (*CString* &a_sFileName, *CString* *a_sRealFileName). This function may be

used by the run-time module 1101 to download cursors 48,027 that should come from the Internet. If this member is NULL, the function is not called.

The Contents of the OBV file

In one embodiment, the data of the project 48,001 may be compressed using the class ECmprs0. According to this embodiment, this class may read a memory area and compress it into another memory area. A compression scheme will now be described. In one embodiment, the memory area is scanned byte by byte and each 0 byte in the input is translated into an 0 bit in the output and each non-zero byte in the input is translated into a 1 bit followed by 8 bits of the actual byte. The end of the output is signed by a 1 bit followed by eight 0 bits (this combination cannot appear inside the output). By way of example, the input: 0, 3, 0, 65, 0, 0, 0, 255, will be translated in the following way:

Input in bytes: 0 3 0 65 0 0 0 255

Output in bits: 0 1000000 11 0 10100 0001 0 0 0 111111111 10000000 0

Output in bytes: 64 212 17 255 128 0

So the output is: 64, 212, 17, 255, 128, 0.

According to this embodiment, the characteristic of a typical project object 48,001 imply that the relative number of zeroes in the input memory area is expected to be very high. Therefore, the size of the output is expected to be smaller than the size of the input.

3.0 Conclusion

The present invention is a system and method for authoring and playing hypervideo 10,007. The present invention permits creating hyperlinks in select regions of interest in frames of a video, defined by hotspots. It is understood that the above description is intended to be illustrative, and not restrictive. Many embodiments will be apparent to those skilled in the art upon reviewing the above description. For example, other implementations of computers, using state machines, parallel processors, client-server architectures, or combinations thereof, may be utilized. Pointing devices, other than mouses, such as pens, joysticks, trackballs and even human fingers, may be used. Displays for televisions and computers, may include, but are not limited to, cathode ray tubes, plasma displays, and liquid crystal displays. The televisions may utilize either analog or digital modulation. The buttons, for example on a mouse, may be any type of switch. Switches may also be implemented with capacitively actuated touch screens. The networks may be computer or broadcast (e.g., television broadcast) networks. Hence, the scope of the invention should, therefore, be determined with reference to the appended claims, along with the full scope of equivalents to which such claims are entitled.

THIS PAGE INTENTIONALLY LEFT BLANK

```
// NSCommands.cpp : implementation file
//

#include "stdafx.h"
#include "stdio.h"
#include "NSCommands.h"

#ifndef PLAYER
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#endif

#ifdef PLAYER
#define ASSERT _ASSERT
#endif

#define ALLOC_BUFFER_SPARE  40//EVEV - add factor
#define ALLOC_HOTSPOTS              3
#define INT_DELIMITER           ' '
#define CHAR_TRUE                   'Y'
#define CHAR_FALSE                  'N'
#define CHAR_EMPTY                  'E'

#define TIME_MULTIPLIER             100.0

RECT y;
```

//

Copyright 1998 Veon, Ltd.

```
//                                  ENSBuffer
////////////////////////////////////////////////////////

ENSBuffer::ENSBuffer( BOOL a_bStore )
{
        m_pBuffer = NULL;
        m_iLen = m_iAlloced = 0;
        m_bStore = a_bStore;
        m_iLoadPtr = 0;
}

ENSBuffer::~ENSBuffer()
{
        Clear();
}

void ENSBuffer::Clear()
{
        if ( m_pBuffer != NULL )
                free ( m_pBuffer );
        m_pBuffer = NULL;
        m_iLen = 0;
        m_iAlloced = 0;
}

BOOL ENSBuffer::Add(const char *a_pBuffer, int a_iLen )
{
        if ( m_iAlloced < m_iLen + a_iLen )
        {
                // Reallocate.
                int l_iAlloced = m_iLen + a_iLen + ALLOC_BUFFER_SPARE;
                char *l_pNewBuffer = (char*)realloc( m_pBuffer,

l_iAlloced*sizeof(char) );
```

Copyright 1998 Veon, Ltd.

```
            if ( l_pNewBuffer == NULL )
                return FALSE;

            m_pBuffer = l_pNewBuffer;
            m_iAlloced = l_iAlloced;
        }

        memcpy( m_pBuffer + m_iLen, a_pBuffer, a_iLen );
        m_iLen += a_iLen;

//      TRACE( "(%.*s)\n", a_iLen, a_pBuffer);

        return TRUE;
}

char *ENSBuffer::GetBuffer( int *a_iLen/*=NULL*/ )
{
        if ( a_iLen != NULL )
            *a_iLen = m_iLen;
        return m_pBuffer;
}

/*BOOL ENSBuffer::SetString( char *a_pString )
{
        Clear();
        return AddString( a_pString );
}
*/
BOOL ENSBuffer::String( ENSString &a_oString )
{
        if ( Storing() )
        {
            if ( a_oString.Get() )
            {
```

Copyright 1998 Veon, Ltd.

```
                int len = ( strlen(a_oString.Get()) );
                Int( len );

                return Add( a_oString.Get(), len );
        }
        else
        {
                char c = CHAR_EMPTY;
                return Add( &c, 1);
        }
}
else
{
        if (m_iLoadPtr >= m_iLen )
        {
                ASSERT(FALSE);
                return FALSE;
        }

        char a_cNext = m_pBuffer[m_iLoadPtr];
        if ( a_cNext == CHAR_EMPTY )
        {
                a_oString.Set( NULL );//EVEV - check
                m_iLoadPtr ++;
        }
        else
        {
                int len;
                Int( len );

                if (m_iLoadPtr+len-1 >= m_iLen )
                {
                        ASSERT(FALSE);
                        return FALSE;
```

Copyright 1998 Veon, Ltd.

```
                }
                a_oString.Set( m_pBuffer+m_iLoadPtr, len );
                m_iLoadPtr += len;
            }
        }
        return TRUE;
}

BOOL ENSBuffer::Bool( BOOL &a_bBool )
{
        if ( Storing() )
        {
            char val = a_bBool? CHAR_TRUE : CHAR_FALSE;
            return Add( &val, 1 );
        }
        else
        {
            if (m_iLoadPtr < m_iLen )
            {
                if ( m_pBuffer[m_iLoadPtr] == CHAR_TRUE )
                {
                    a_bBool = TRUE;
                }
                else if ( m_pBuffer[m_iLoadPtr] == CHAR_FALSE )
                {
                    a_bBool = FALSE;
                }
                else
                {
                    ASSERT(FALSE);
                    return FALSE;
                }

                m_iLoadPtr++;
```

Copyright 1998 Veon, Ltd.

```
                return TRUE;
            }
            ASSERT(FALSE);
            return FALSE;
        }
}

BOOL ENSBuffer::Int( int &a_iInt )
{
        if ( Storing() )
        {
            char buf[10];
            sprintf( buf, "%d%c", a_iInt, INT_DELIMITER );
            return Add( buf, strlen(buf) );
        }
        else
        {
            int l_iSign = 1;
            a_iInt = 0;
            for ( ;m_iLoadPtr < m_iLen && m_pBuffer[m_iLoadPtr] !=
INT_DELIMITER;
                m_iLoadPtr ++       )
            {
                if ( m_pBuffer[m_iLoadPtr] == '-' &&
                    a_iInt == 0 )
                {
                    // Minus sign allowed on the beginning.
                    l_iSign = -1;
                }
                else
                {
                    if ( m_pBuffer[m_iLoadPtr] < '0' ||
                        m_pBuffer[m_iLoadPtr] > '9' )
                    {
```

Copyright 1998 Veon, Ltd.

```
                                ASSERT(FALSE);
                                return FALSE;
                            }
                            a_iInt = a_iInt * 10 + m_pBuffer[m_iLoadPtr] - '0';
                        }
                    }
                    a_iInt *= l_iSign;

                    m_iLoadPtr ++;          // Delimiter
                }
                return TRUE;
}

BOOL ENSBuffer::Long( long &a_lLong )
{
        int temp = a_lLong;
        BOOL l_bRC = Int( temp );
        a_lLong = temp;

        return l_bRC;
}

////////////////////////////////////////////////////////
//                          ENSString
////////////////////////////////////////////////////////

ENSString::ENSString()
```

Copyright 1998 Veon, Ltd.

```
        : ENSBuffer( TRUE )
{
}

ENSString::~ENSString()
{
}

BOOL ENSString::Set( const char *a_pString )
{
        Clear();
        if ( a_pString )
                return Add( a_pString, strlen(a_pString) + 1 );

        return TRUE;
}

BOOL ENSString::Set( const char *a_pString, int a_iLen )
{
        Clear();
        if ( a_pString )
        {
                Add( a_pString, a_iLen );
                return Add( "", 1 );
        }
        return TRUE;
}

char *ENSString::Get()
{
        return GetBuffer();
}
```

Copyright 1998 Veon, Ltd.

```
//////////////////////////////////////////////////////////
//                          ENSFrameInfoManager
//////////////////////////////////////////////////////////

ENSFrameInfoManager::ENSFrameInfoManager()
{
        m_pInfo = NULL;
}
```

Copyright 1998 Veon, Ltd.

```
ENSFrameInfoManager::~ENSFrameInfoManager()
{
        if ( m_pInfo != NULL )
                delete m_pInfo;
}

BOOL ENSFrameInfoManager::NewInfo( ENSFrameInfo *a_pNewInfo,
ENSBuffer &a_oBuffer)
{
        // Here we may pass only the difference!

        if (m_pInfo != NULL )
                delete m_pInfo;

        m_pInfo = a_pNewInfo;

        // Create the command.
        a_oBuffer.Clear();

        // write command version.
        int l_iVersion = VANS_CMD_VERSION;
        a_oBuffer.Int( l_iVersion );

        if (!m_pInfo->Serial( a_oBuffer ) )
                return FALSE;

        // Make it null terminated.
        a_oBuffer.Add("", 1);

        return TRUE;
}

BOOL ENSFrameInfoManager::NewCommand( char *a_pCommand )
{
```

Copyright 1998 Veon, Ltd.

```
        ENSBuffer l_oBuffer( FALSE/*Load*/ );

        l_oBuffer.Add( a_pCommand, strlen( a_pCommand ) );

        if (m_pInfo != NULL )
        {
                delete m_pInfo;
                m_pInfo = NULL;
        }

        // Read command version.
        int l_iVersion;
        if ( !l_oBuffer.Int( l_iVersion ) ||
                 l_iVersion != VANS_CMD_VERSION      )
        {
                // Wrong version.
                return FALSE;
        }

        m_pInfo = new ENSFrameInfo;

        return m_pInfo->Serial( l_oBuffer );
}

void ENSFrameInfoManager::Clear()
{
        if (m_pInfo != NULL )
        {
                delete m_pInfo;
                m_pInfo = NULL;
        }
}
```

Copyright 1998 Veon, Ltd.

```
//////////////////////////////////////////////////////////
//                          ENSFrameInfo
//////////////////////////////////////////////////////////

BOOL ENSFrameInfo::Serial( ENSBuffer &a_oBuffer )
{
        // event
        BOOL l_bEvent = ( m_pEvent != NULL );
        a_oBuffer.Bool( l_bEvent );
        if ( l_bEvent )
```

Copyright 1998 Veon, Ltd.

```
{
        if ( a_oBuffer.Loading() )
        {
                // We have to allocate
                m_pEvent = new ENSEvent(NULL);
        }
        m_pEvent->Serial( a_oBuffer );
}

// Hotspots count
a_oBuffer.Int( m_iHotspotsCount );

// Hotspots
if ( a_oBuffer.Loading() )
{
        // We have to allocate the hotspots array.
        ASSERT( m_aHotspots == NULL );

        m_iHotspotsAlloced = m_iHotspotsCount;
        m_aHotspots = (ENSHotspot**) malloc(
m_iHotspotsAlloced*sizeof(ENSHotspot*) );
        if ( m_aHotspots == NULL )
                return FALSE;
}

for (int i = 0; i < m_iHotspotsCount; i++ )
{
        if ( a_oBuffer.Loading() )
                m_aHotspots[i] = new
ENSHotspot(NULL,NULL,NULL);

        m_aHotspots[i]->Serial( a_oBuffer );
}

return TRUE;//EVEV - check!!
```

Copyright 1998 Veon, Ltd.

```
}

ENSFrameInfo::ENSFrameInfo()
{
        m_pEvent = NULL ;
        m_iHotspotsCount = 0;
        m_iHotspotsAlloced = 0;
        m_aHotspots = NULL;
}

ENSFrameInfo::~ENSFrameInfo()
{
        Clear();
}

void ENSFrameInfo::Clear()
{
        SetEvent( NULL );

        if ( m_aHotspots != NULL )
        {
                for ( int i = 0; i < m_iHotspotsCount; i++ )
                {
                        if ( m_aHotspots[i] != NULL )
                                delete m_aHotspots[i];
                }
                free( m_aHotspots );
                m_aHotspots = NULL;
        }

        m_iHotspotsCount = 0;
        m_iHotspotsAlloced = 0;
}
```

Copyright 1998 Veon, Ltd.

```
void ENSFrameInfo::SetEvent( ENSEvent *a_pEvent )
{
        if ( m_pEvent != NULL )
                delete m_pEvent;

        m_pEvent = a_pEvent;
}

BOOL ENSFrameInfo::AddHotspot( ENSHotspot *a_pHotspot )
{
        if ( a_pHotspot == NULL )
        {
                ASSERT(FALSE);
                return FALSE;
        }

        if ( m_iHotspotsAlloced <= m_iHotspotsCount )
        {
                // Reallocate.
                int l_iHotspotsAlloced = m_iHotspotsCount + 1 +
ALLOC_HOTSPOTS;
                ENSHotspot** l_aNewHotspots = (ENSHotspot**)
                                                realloc( m_aHotspots,

l_iHotspotsAlloced*sizeof(ENSHotspot*) );
                if ( l_aNewHotspots == NULL )
                        return FALSE;

                m_aHotspots = l_aNewHotspots;
                m_iHotspotsAlloced = l_iHotspotsAlloced;
        }

        m_aHotspots[ m_iHotspotsCount ] = a_pHotspot;
        m_iHotspotsCount ++;
        return TRUE;
```

Copyright 1998 Veon, Ltd.

```
}

//////////////////////////////////////////////////////////
//                          ENSFrameLinkData
//////////////////////////////////////////////////////////
ENSFrameLinkData::ENSFrameLinkData(const char *a_pName)
{
        m_sName.Set( a_pName );
        m_eType = LINK;
        m_bURL = FALSE;
        m_bASF = FALSE;
        m_bSeekTime = FALSE;
        m_bSeekMarker = FALSE;
```

Copyright 1998 Veon, Ltd.

```
        m_dSeekTime = 0.0;
        m_bText = FALSE;
}


BOOL ENSFrameLinkData::Serial( ENSBuffer &a_oBuffer )
{
        a_oBuffer.String( m_sName );

        int temp = (int) m_eType;
        a_oBuffer.Int ( temp );
        m_eType = (EType)temp;      // If storing -> temp didn't changed

        if ( m_eType == LINK )
        {
                a_oBuffer.Bool( m_bURL );
                if ( m_bURL )
                {
                        a_oBuffer.String( m_sURL );
                        a_oBuffer.String( m_sURLFrame );
                        a_oBuffer.String( m_sURLLocation );
                }

                a_oBuffer.Bool( m_bASF );
                if ( m_bASF )
                        a_oBuffer.String( m_sASF );

                a_oBuffer.Bool( m_bSeekTime );
                if ( m_bSeekTime )
                {
                        int time100;    // Store time multiplied by 100

                        if ( a_oBuffer.Storing() )
                                time100 = m_dSeekTime*TIME_MULTIPLIER;
```

Copyright 1998 Veon, Ltd.

```
                a_oBuffer.Int( time100 );

                if ( a_oBuffer.Loading() )
                        m_dSeekTime =
((double)time100)/TIME_MULTIPLIER;
        }

        a_oBuffer.Bool( m_bSeekMarker );
        if ( m_bSeekMarker )
                a_oBuffer.String( m_sSeekMarker );

        a_oBuffer.Bool( m_bText );
        if ( m_bText )
                a_oBuffer.String( m_sText );
    }

    return TRUE;  //EVEV-check!!!
}

////////////////////////////////////////////////////////
//                      ENSEvent
////////////////////////////////////////////////////////

////////////////////////////////////////////////////////
//                      ENSHotspot
////////////////////////////////////////////////////////

ENSHotspot::ENSHotspot( const char *a_pName,
                        const char *a_pDescription,
                            ENSGeometry *a_pGeometry )
                                                :
ENSFrameLinkData (a_pName)
{
    m_sDescription.Set( a_pDescription );
```

Copyright 1998 Veon, Ltd.

```
        m_pGeometry = a_pGeometry;
        m_lMoveCursorID = -1;       // No cursor
        m_lClickCursorID = -1;      // No cursor
}

ENSHotspot::~ENSHotspot()
{
        if ( m_pGeometry != NULL )
                delete m_pGeometry;
}

BOOL ENSHotspot::Serial( ENSBuffer &a_oBuffer )
{
        ENSFrameLinkData::Serial( a_oBuffer );

        // Description
        a_oBuffer.String( m_sDescription );

        // Cursors
        a_oBuffer.Long( m_lMoveCursorID );
        a_oBuffer.Long( m_lClickCursorID );

        // Geometry type.
        if ( a_oBuffer.Storing() )
        {
                // Geometry must exist.
                if ( m_pGeometry == NULL )
                {
                        ASSERT(FALSE);
                        return FALSE;
                }

                // Store type
                int temp = (int)m_pGeometry->Type();
```

Copyright 1998 Veon, Ltd.

```
        a_oBuffer.Int ( temp );
    }
    else
    {
        // Get type
        int temp;
        a_oBuffer.Int ( temp );
        ENSGeometry::EType l_eType = (ENSGeometry::EType)temp;

        // We have to allocate the geometry
        switch (l_eType)
        {
            case ENSGeometry::ELLIPSE:
                POINT pt; pt.x=pt.y=0;
                m_pGeometry = new ENSEllipse(pt,0,0);
                break;
            case ENSGeometry::RECTANGLE:
                RECT rc; rc.left=rc.top=rc.right=rc.bottom=0;
                m_pGeometry = new ENSRect(rc);
                break;
            case ENSGeometry::POLYGON:
                m_pGeometry = new ENSPolygon(0);
                break;
            default:
                ASSERT(FALSE);
                return FALSE;
        }
    }

    return m_pGeometry->Serial( a_oBuffer );
}
```

Copyright 1998 Veon, Ltd.

```
////////////////////////////////////////////////////////
//                          ENSGeometry
////////////////////////////////////////////////////////

ENSGeometry::ENSGeometry( EType a_eType )
{
        m_eType = a_eType;
}

ENSGeometry::~ENSGeometry()
{
}
```

Copyright 1998 Veon, Ltd.

```
// Normalize rect.
void ENSGeometry::NormalizeRect( RECT &a_oRect)
{
        if ( a_oRect.right < a_oRect.left )
        {
                long temp = a_oRect.right;
                a_oRect.right = a_oRect.left;
                a_oRect.left = temp;
        }
        if ( a_oRect.bottom < a_oRect.top )
        {
                long temp = a_oRect.bottom;
                a_oRect.bottom = a_oRect.top ;
                a_oRect.top = temp;
        }
}



////////////////////////////////////////////////////////
//                          ENSEllipse
////////////////////////////////////////////////////////

ENSEllipse::ENSEllipse( POINT a_oCenter, long a_lVRadius, long a_lHRadius)
: ENSGeometry ( ENSGeometry::ELLIPSE )
{
        m_oCenter = a_oCenter;
        m_lVRadius = a_lVRadius;
        m_lHRadius = a_lHRadius;
}

BOOL ENSEllipse::Serial( ENSBuffer &a_oBuffer )
{
```

Copyright 1998 Veon, Ltd.

```
        a_oBuffer.Long ( m_oCenter.x );
        a_oBuffer.Long ( m_oCenter.y );
        a_oBuffer.Long ( m_lVRadius );
        a_oBuffer.Long ( m_lHRadius );

        return TRUE;//EVEV-check!!
}

BOOL ENSEllipse::PtInside( POINT& a_oPoint )
{
        // The ellipse is a line or a point.
        if ( m_lHRadius == 0 || m_lVRadius == 0 )
                return FALSE;

        // Calculate DeltaX and DeltaY
        float x = (float)a_oPoint.x - (float)m_oCenter.x ;
        float y = (float)a_oPoint.y - (float)m_oCenter.y ;

        // Check if the point is inside the ellipse.
        if ((((x/m_lHRadius)*(x/m_lHRadius)) +
((y/m_lVRadius)*(y/m_lVRadius))) <= 1.0)
                return TRUE ;
        else
                return FALSE ;
}
```

Copyright 1998 Veon, Ltd.

```
////////////////////////////////////////////////////////
//                          ENRect
////////////////////////////////////////////////////////

ENSRect::ENSRect( RECT& a_oRect )
: ENSGeometry ( RECTANGLE)
{
        m_oRect = a_oRect;
        NormalizeRect( m_oRect );
}

BOOL ENSRect::Serial( ENSBuffer &a_oBuffer )
{
        a_oBuffer.Long ( m_oRect.left );
        a_oBuffer.Long ( m_oRect.top );
```

Copyright 1998 Veon, Ltd.

```
        a_oBuffer.Long ( m_oRect.right );
        a_oBuffer.Long ( m_oRect.bottom );

        return TRUE;//EVEV-check!!
}

BOOL ENSRect::PtInside( POINT& a_oPoint )
{
        return PtInRect( &m_oRect, a_oPoint );
}
```

```
////////////////////////////////////////////////////////
//                          ENSTriangle
////////////////////////////////////////////////////////

/*ENSTrianlge::ENSTrianlge( POINT& a_oCenter, long a_lVRadius, long
a_lHRadius)
: ENSGeometry ( ENSGeometry::ELLIPSE )
{
        m_oCenter = a_oCenter;
        m_lVRadius = a_lVRadius;
        m_lHRadius = a_lHRadius;
}

BOOL ENSTrianlge::Serial( ENSBuffer &a_oBuffer )
{
        a_oBuffer.Long ( m_oCenter.x );
```

Copyright 1998 Veon, Ltd.

```
        a_oBuffer.Long ( m_oCenter.y );
        a_oBuffer.Long ( m_lVRadius );
        a_oBuffer.Long ( m_lHRadius );

        return TRUE;//EVEV-check!!
}

BOOL ENSTrianlge::PtInside( POINT& a_oPoint )
{
        // The ellipse is a line or a point.
        if ( m_lHRadius == 0 || m_lVRadius == 0 )
                return FALSE;

        // Calculate DeltaX and DeltaY
        float x = (float)a_oPoint.x - (float)m_oCenter.x ;
        float y = (float)a_oPoint.y - (float)m_oCenter.y ;

        // Check if the point is inside the ellipse.
        if ((((x/m_lHRadius)*(x/m_lHRadius)) +
((y/m_lVRadius)*(y/m_lVRadius))) <= 1.0)
                return TRUE ;
        else
                return FALSE ;
}
*/
```

Copyright 1998 Veon, Ltd.

```
//////////////////////////////////////////////////////////
//                          ENSPolygon
//////////////////////////////////////////////////////////

ENSPolygon::ENSPolygon(long a_lNumOfPoints )
: ENSGeometry ( ENSGeometry::POLYGON )
{
        m_lNumOfPoints = a_lNumOfPoints;
        if (m_lNumOfPoints > 0)
        {
                m_aPoints = new POINT[m_lNumOfPoints];
        }
        else
                m_aPoints = NULL;
}
```

Copyright 1998 Veon, Ltd.

```
ENSPolygon::~ENSPolygon()
{
        if (m_aPoints !=NULL)
                delete m_aPoints;
}

BOOL ENSPolygon::Serial( ENSBuffer &a_oBuffer )
{
        a_oBuffer.Long ( m_lNumOfPoints );

        if (a_oBuffer.Loading() )
        {
                if (m_aPoints !=NULL)
                        delete m_aPoints;

                m_aPoints = new POINT[m_lNumOfPoints];
        }

        for (int i= 0; i< m_lNumOfPoints; i++)
        {
                a_oBuffer.Long( m_aPoints[i].x);
                a_oBuffer.Long( m_aPoints[i].y);
        }

        return TRUE;//EVEV-check!!
}

BOOL ENSPolygon::PtInside( POINT& a_oPoint )
{
                // Implement the algorithm for checking if a point is inside a
polygon.

        HRGN l_hRgn = CreatePolygonRgn ( m_aPoints, m_lNumOfPoints,
WINDING);
```

Copyright 1998 Veon, Ltd.

```
        BOOL l_bResult = ( PtInRegion ( l_hRgn, a_oPoint.x, a_oPoint.y ));

        DeleteObject( l_hRgn );

        return l_bResult;
}

void ENSPolygon::SetPoint(long a_lIndex, POINT& a_oPoint)
{
        m_aPoints[a_lIndex] = a_oPoint;
}
```

Copyright 1998 Veon, Ltd.

```
// NSCommands.h : header file
//

#ifndef __NS_COMMANDS_H__
#define __NS_COMMANDS_H__
```

Copyright 1998 Veon, Ltd.

```
////////////////////////////////////////////////////////
//                          ENSBuffer
////////////////////////////////////////////////////////

enum ENSCommandVersion
{
        VANS_CMD_VER_UNKNOWN = -1,
        VANS_CMD_VER_BETA1   = 0,
};

#define VANS_CMD_VERSION     VANS_CMD_VER_BETA1

class ENSString;

class ENSBuffer
{
public:
        ENSBuffer( BOOL a_bStore );
        virtual ~ENSBuffer();
        void ENSBuffer::Clear();
        BOOL Add( const char *a_pBuffer, int a_iLen );
        char *GetBuffer( int *a_iLen=NULL );

//      BOOL SetString( char *a_pString );
        BOOL String( ENSString &a_oString );
//      char *GetString() { return (char *)m_pBuffer; }//EVEV-delete
        BOOL Int( int &a_iInt );
        BOOL Bool( BOOL &a_bBool );
        BOOL Long( long &a_lLong );

        BOOL Storing() { return m_bStore; }
        BOOL Loading() { return !m_bStore; }

private:
```

Copyright 1998 Veon, Ltd.

```
        BOOL m_bStore;
        char *m_pBuffer;
        int m_iLen;
        int m_iAlloced;
        int m_iLoadPtr;
};

class ENSString : private ENSBuffer
{
public:
        ENSString();
        virtual ~ENSString();

        BOOL Set( const char *a_pString );
        BOOL Set( const char *a_pString, int a_iLen );
        char *Get();
};
```

```
////////////////////////////////////////////////////////
//                          ENSGeometry
```

Copyright 1998 Veon, Ltd.

```
//////////////////////////////////////////////////////
class ENSGeometry
{
public:
        enum EType
        {
                UNDEFINED,
                ELLIPSE ,           // Defines an elliptic Geometry
                TRIANGLE ,          // Triangular Geometry.
                RECTANGLE ,                 // Rectangular Geometry.
                POLYGON             // Polygon Geometry. Holds irregular ROIs.
        };

        ENSGeometry( EType a_eType );
        virtual ~ENSGeometry();

        virtual BOOL PtInside( POINT& a_oPoint ) = 0;
        virtual BOOL Serial( ENSBuffer &a_oBuffer ) = 0;
        static void NormalizeRect( RECT &a_oRect);

        EType Type() { return m_eType; }

private:
        EType m_eType;
};


//////////////////////////////////////////////////////
//                          ENSEllipse
//////////////////////////////////////////////////////
class ENSEllipse : public ENSGeometry
{
public:
        POINT m_oCenter;
```

Copyright 1998 Veon, Ltd.

```
        long m_lVRadius;
        long m_lHRadius;

        ENSEllipse( POINT a_oCenter, long a_lVRadius, long a_lHRadius);
        virtual BOOL PtInside( POINT& a_oPoint );
        virtual BOOL Serial( ENSBuffer &a_oBuffer );
};
```

```
//////////////////////////////////////////////////////////
//                          ENSRect
//////////////////////////////////////////////////////////
class ENSRect : public ENSGeometry
```

Copyright 1998 Veon, Ltd.

```
{
public:
        RECT m_oRect;

        ENSRect( RECT& a_oRect );
        virtual BOOL PtInside( POINT &a_oPoint );
        virtual BOOL Serial( ENSBuffer &a_oBuffer );
};

////////////////////////////////////////////////////////
//                              ENSPolygon
////////////////////////////////////////////////////////
class ENSPolygon : public ENSGeometry
{
public:
        long m_lNumOfPoints;
        POINT *m_aPoints;

        ENSPolygon( long a_lNumOfPoints);
        virtual ~ENSPolygon();

        virtual BOOL PtInside( POINT& a_oPoint );
        virtual BOOL Serial( ENSBuffer &a_oBuffer );
        void SetPoint(long a_lIndex, POINT& a_oPoint);
};




////////////////////////////////////////////////////////
//                              ENSFrameLinkData
////////////////////////////////////////////////////////

class ENSFrameLinkData
```

Copyright 1998 Veon, Ltd.

```
{
public:
        ENSString m_sName;

        enum EType
        {
                LINK,
                BACK,
                EXIT,
        };

        EType m_eType;

        BOOL m_bURL;
        ENSString m_sURL;
        ENSString m_sURLFrame;
        ENSString m_sURLLocation;

        BOOL m_bASF;
        ENSString m_sASF;

        BOOL m_bSeekTime;
        double m_dSeekTime;

        BOOL m_bSeekMarker;
        ENSString m_sSeekMarker;

        BOOL m_bText;
        ENSString m_sText;

        ENSFrameLinkData( const char *a_pName );
        virtual BOOL Serial( ENSBuffer &a_oBuffer );
};
```

Copyright 1998 Veon, Ltd.

```
////////////////////////////////////////////////////////////
//                                        ENSEvent
////////////////////////////////////////////////////////////
#define ENSEvent ENSFrameLinkData
```

Copyright 1998 Veon, Ltd.

```
//////////////////////////////////////////////////////////
//                          ENSHotspot
//////////////////////////////////////////////////////////
class ENSHotspot : public ENSFrameLinkData
{
public:

        ENSHotspot( const char *a_pName, const char *a_pDescription,
ENSGeometry *a_pGeometry );
        virtual ~ENSHotspot();

        virtual BOOL Serial( ENSBuffer &a_oBuffer );

//EVEV do we need this?     char *m_pName;
        ENSString       m_sDescription;
        ENSGeometry             *m_pGeometry;
        long            m_lMoveCursorID;
        long            m_lClickCursorID;
};
```

```
//////////////////////////////////////////////////////////
//                          ENSFrameInfo
//////////////////////////////////////////////////////////
class ENSFrameInfo
{
public:
        ENSFrameInfo();
```

Copyright 1998 Veon, Ltd.

```
        virtual ~ENSFrameInfo();
        void Clear();

        void SetEvent( ENSEvent *a_pEvent );
        BOOL AddHotspot( ENSHotspot *a_pHotspot );
        virtual BOOL Serial( ENSBuffer &a_oBuffer );

        ENSEvent *Event() { return m_pEvent; }
        int      HotspotsCount() { return m_iHotspotsCount; }
        ENSHotspot **Hotspots() { return m_aHotspots; }

private:

        // The version of the command.
        int              m_iVersion;

        ENSEvent *m_pEvent;
        int      m_iHotspotsCount;
        int      m_iHotspotsAlloced;
        ENSHotspot **m_aHotspots;
};


//////////////////////////////////////////////////////////
//                              ENSFrameInfoManager
//////////////////////////////////////////////////////////

class ENSFrameInfoManager
{
public:
        ENSFrameInfoManager();
        virtual ~ENSFrameInfoManager();

        BOOL NewCommand( char *a_pCommand );
```

Copyright 1998 Veon, Ltd.

```
        BOOL NewInfo( ENSFrameInfo *a_pNewInfo, ENSBuffer
&a_oNSBuffer);

        void Clear();

        ENSFrameInfo *Info() { return m_pInfo; }

private:
        ENSFrameInfo *m_pInfo;
};



#endif
```

Copyright 1998 Veon, Ltd.

```
// stdafx.h : include file for standard system include files,

//  or project specific include files that are used frequently, but

//      are changed infrequently

//


#include <afxwin.h>         // MFC core and standard components

#include <afxext.h>         // MFC extensions
```

Copyright 1998 Veon, Ltd.

```
#if !defined(__hvclpfile_h)
#define __hvclpfile_h
```

Copyright 1998 Veon, Ltd.

```
/* HV Project
        Ephyx Technologies Ltd.
        Copyright © 1994. All Rights Reserved.

        SUBSYSTEM:   hvtest.exe Application
        FILE:        hvclp.h
        AUTHOR:      Avner Peleg


        OVERVIEW
        ========
        Class definition for a clip file.
*/


#include <stdio.h>
#include <string.h>

#include <iostream.h>
#include <fstream.h>

// general macros

#define HV_OK 0
#define HV_ERROR -1


enum HvVideoTypes { HV_AVI , HV_MPEG , HV_JPEG , HV_CINEPAC ,
HV_INDEO ,  HV_QUICKTIME } ;

class HvClipMarker {    // a structure to hold a marker for a clip
 public:
  long mark ;    // time stamp in frames
  char *id ;     // marker's id
```

Copyright 1998 Veon, Ltd.

```
 HvClipMarker() { id = 0 ; }
 HvClipMarker(long fr , char *name) { mark = fr ; id = ((name == 0 ) ? 0 :
strdup(name)) ; }
 HvClipMarker( HvClipMarker& sp) { mark = sp.mark ; id = ((sp.id == 0 ) ? 0 :
strdup(sp.id)) ; }
} ;

class HvClipROI {
 public:
  int x ;
  int y ;
  int a ;
  int b ;
  float theta ;
  long id ; // id of the sequence.
  class HvClipROI* next ;
  class HvClipROI* prev ;
  class HvClipROI* nextInSequence ;
  HvClipROI() { id = 0 ; nextInSequence = 0 ; }
  HvClipROI(long handle, int x, int y, int a, int b, float theta, HvClipROI *n,
HvClipROI *p)
                                                : x(x) , y(y) , nextInSequence(0) ,
                                                    a(a) , b(b) , theta(theta) ,
next(n), prev(p) , id(handle) {}
  virtual long PtInROI(const int x , const int y ) const ;
} ;

class HvSequenceIterator ;

class HvSequence {
  friend class HvSequenceIterator ;
 public:
  HvSequence() {name = 0 ; id = 0 ; first = last = 0 ;}
  HvSequence(long ,char*, HvClipROI* , HvClipROI* ) ;
  long id ;
```

Copyright 1998 Veon, Ltd.

```
 char* name ;
 HvClipROI *first ;
 HvClipROI *last ;
} ;

class HvSequenceIterator {
 HvClipROI* current ;
 HvSequence* seq ;
 public:
        HvSequenceIterator() {current =0 ;seq =  0 ;}
        HvSequenceIterator(HvSequence*) ;
        HvClipROI* Next() ;
        HvClipROI* First() ;
        HvClipROI* Last() ;
} ;


class HvClipStruct {
// debug
public:
        char *_name ;                   // identifier of the clip in the project
        char *_mediaFilePath ; //  full path of the video data file
        HvVideoTypes _type ;      //  type of video : AVI,etc..

        int _numMarkers ;               //   the number of markers defined
        int _numAllocatedMarkers ; // the size of the allocated list
        HvClipMarker* _listOfMarkers ; // the list of markers defined for the clip

        int maxFramesForRoi ;  // number of allocated frame slots for ROIS;
        HvClipROI** _listOfRoiByFrame ;  // a list of ROIs for each frame of the
clip

        int allocFrameSlots(const long) ;  // allocate new frame slots if needed
        int allocMarkerSlots() ;                // allocate new marker slots if
needed
```

Copyright 1998 Veon, Ltd.

```
        int allocSequenceSlots() ;              // allocate new sequence slots if
needed

        long GetSequenceId() ;                          // derive an ID for a
sequence
        int IdExists(const int ) ;    // checks whether a sequence with the given id
exists

        HvSequence* _bagOfSequences ;  // list of all sequences defined for the
clip
        int _numSequences ;                             // number of
sequences
        int _numAllocatedSequences ;    // number of allocated sequence slots

 public:
        HvClipStruct() : _type(HV_AVI) {}  // empty ctor
        HvClipStruct(int &rc, const char *name, HvVideoTypes type) ; // full
ctor for definition
        HvClipStruct(int& rc, const char *filePath) ; // full ctor from file
        virtual ~HvClipStruct() ;  // dtor

        void Write(ostream& os) ;  // a write method
        int Write(const char *filePath) ; // a write method into a file

        void Dump(ostream& os) ;  // a dump method
        int Dump(const char *filePath) ; // a dump method into a file

        // get & set the clip name
        char *name() { return _name ; }
        void setName(const char *name)  ;

        // get & set the clip video file path
        char *mediaFile() { return _mediaFilePath ; }
        void setMediaFile(const char *name)  ;

        // add a sequence
```

Copyright 1998 Veon, Ltd.

```
        int AddClipSequence(const char* seqName) ;
        // delete a sequence
        int DelSequence(const char* seqName) ;
        int DelSequence(long seqId) ;

        HvClipROI* GetRoiOfFrame(long fr) { if (fr < maxFramesForRoi) return
_listOfRoiByFrame[fr] ;
                        else return 0 ; }
        char* GetSequence(long seqId) ;

        int GetSequences(char ***names) ;
        int GetSequencesExt(char ***names) ;

        void AddToSequence(HvClipROI* ); // adds a ROI to a sequences
(always at the end)
        // add a ROI stamp
        int addRoi( long t,long handle, int x, int y, int a, int b, float theta = 0.0 );

        // add a marker on the clip
        int addMarker( long t,char *id );

        // delete a marker
        int delMarker(char *id) {return 1;}

        // get an array of markers
        int getMarkers(long *ts, char **ids) {return 1;}

        // find the object at (t,x,y) of the clip
        long ObjectAtFramePoint(const long t,const int x, const int y) const ;

        // clear object's references
        int delObjectRefs(const long handle) {return 1;}
};
```

Copyright 1998 Veon, Ltd.

```
#endif
#include "stdafx.h"
#include <string.h>
#include "hvflstr.h"

BOOL EGetStr( FILE* f, char* s, int max_s )
{
        WORD l;

        // get the line length
        if( fread(&l, sizeof(WORD), 1, f) != 1 )
                return FALSE;

        if( l <= 0 || l+1 > max_s )
                return FALSE;

        // read the line itself
        if( fread(s, 1, l, f) != l )
                return FALSE;

        for( int i = 0; i < l-1; i++, s++ )
                *s = *s ^ 255;
        *s = '\0';

        return TRUE;
}


BOOL EPutStr( FILE* f, char* s )
{
        WORD l = strlen(s);

        // write the line length
        if( fwrite(&l, sizeof(WORD), 1, f) != 1 )
```

Copyright 1998 Veon, Ltd.

```
            return FALSE;

        char *p = s;
        for( int i = 0; i < l; i++, p++ )
            *p = *p ^ 255;

        // write the line itself
        if( fwrite(s, 1, l, f) != l )
            return FALSE;

        return TRUE;
}
```

Copyright 1998 Veon, Ltd.

```
#ifndef __EFLCSTR__H_
#define __EFLCSTR__H_

#include "stdio.h"

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

typedef int BOOL;

typedef unsigned short WORD;                    // 16 bits long

BOOL EGetStr( FILE* f, char* s, int max_s ) ;

BOOL EPutStr( FILE* f, char* s ) ;

#endif
```

Copyright 1998 Veon, Ltd.

```
/*  HV Project
        Ephyx Technologies Ltd.
        Copyright © 1994. All Rights Reserved.

        SUBSYSTEM:    hvtest.exe Application
        FILE:         hvobj.cpp



        OVERVIEW
        ========
        Class member methods for an object file.



        file structure:
        <object name>
        <num sequences>
                        <sequence>      each sequence is <clip name> ,
<identifier>
*/

#include "stdafx.h"
#include "hvobj.h"
#include "hvflstr.h"

static char flstr[256];

////////////////////////  HvObjSequence  ////////////////////////////////////
//
// definition ctor
//
HvObjSequence::HvObjSequence(const char* clip, const char* seq,
HvObjSequence* n) :
```

Copyright 1998 Veon, Ltd.

```
        next(n)
{
 if (clip != 0)
        clipName = strdup(clip) ;

 if (seq != 0)
        clipSequence = strdup(seq) ;
}

////////////////////////////  HvObjectStruct      ////////////////////////////////
//
// definition ctor
//
HvObjectStruct::HvObjectStruct(const char *id)
{

 numSequences = 0 ;
 theSequencesListHeader = 0 ;
 theTarget = 0 ;

 // copy the name
 if (id != 0)
        {
         this->id = strdup(id);
        }
 else
         id = "no_name" ;
}

//
// a ctor from a file path:
//   opens the file and reads the info
//   rc = -1 if fails
HvObjectStruct::HvObjectStruct( int &rc, const char *filePath )
```

Copyright 1998 Veon, Ltd.

```
{
        FILE *fileDesc ;
        char line[256] ;
        char line2[256] ;


        rc = HV_OK ;

        if (filePath != 0 && filePath[0] != '\0')
                {
                // open the file for read only
                if ((fileDesc = fopen(filePath, "rb")) == 0 )
                        {
                                rc = HV_ERROR ;
                                return ;
                        }

                // get the name of the Object
                if (!EGetStr(fileDesc,line,255))
                        {
                                rc = HV_ERROR ;
                                fclose(fileDesc) ;
                                return ;
                        }
                // allocate and copy the data
                if ((id  = new char[strlen(line) + 1]) == 0)
                        {
                                rc = HV_ERROR ;
                                fclose(fileDesc) ;
                                return ;
                        }
                strcpy(id , line) ;

                // read the sequences set for the Object
```

Copyright 1998 Veon, Ltd.

```
if (!EGetStr(fileDesc,line,255))
        {
                rc = HV_ERROR ;
                fclose(fileDesc) ;
                return ;
        }

FILE* fd = fopen("c:\\temp\\moshe.log","a+") ;
fprintf(fd,"%s",line) ;
fclose(fd) ;

sscanf(line,"%d",&numSequences) ;
theSequencesListHeader = 0 ;
// loop and read all the sequences
for ( int i = 0 ; i < numSequences ; i++)
        {
                if ( !EGetStr(fileDesc,flstr,255))
                        {
                                rc = HV_ERROR ;
                                fclose(fileDesc) ;
                                return ;
                        }
                sscanf(flstr, "%s %s", line2, line) ;
                theSequencesListHeader = new
HvObjSequence(line2,line,

theSequencesListHeader) ;
        }

// get the target
int isTargeted ;
EGetStr(fileDesc,flstr,255) ;
sscanf(flstr,"%d",&isTargeted);
if (isTargeted != 0)
```

Copyright 1998 Veon, Ltd.

```
                    {
                     theTarget = new char[isTargeted+4];
                     strcpy(theTarget, flstr);
                     while (*theTarget != ' ') theTarget++ ;
                     while (*theTarget == ' ') theTarget++ ;
                     theTarget[strlen(theTarget)] = '\0' ;
                    }
                else
                     theTarget = 0 ;

                 fclose(fileDesc) ;
                }
        else
                {
                        rc = HV_ERROR ;
                }

        return ;
}

//
// a virtual dtor
//
HvObjectStruct::~HvObjectStruct()
{
        HvObjSequence* temp;

        if (id != 0)
          delete id ;

        if (theTarget != 0)
          delete theTarget ;

        if (theSequencesListHeader != 0)
```

Copyright 1998 Veon, Ltd.

```
        while ((temp = theSequencesListHeader->next) != 0)
                {
                        delete theSequencesListHeader ;
                        theSequencesListHeader = temp ;
                }
}

int HvObjectStruct::AddObjSequence(const char* clip, const char* seq)
{
 return AddObjSequence(new HvObjSequence(clip,seq,0));
}

int HvObjectStruct::ExistObjSequence(const char* clip, const char* seq)
{
        HvObjSequence* temp = theSequencesListHeader ;
        while (temp != 0)
         {
                if ((strcmp(temp->clipName, clip) == 0) &&
                        (strcmp(temp->clipSequence, seq)  == 0))
                                return 1 ;
                temp = temp->next ;
         }

        return 0 ;
}

int HvObjectStruct::AddObjSequence(HvObjSequence* seq)
{
 if (seq == 0)
        return HV_ERROR ;

 seq->next  = theSequencesListHeader ;
 theSequencesListHeader = seq ;
```

Copyright 1998 Veon, Ltd.

```
 numSequences++ ;

 return HV_OK ;
}

int HvObjectStruct::GetSequences(HvObjSequence** theSequences)
{
 *theSequences = theSequencesListHeader ;
 return numSequences ;
}

int HvObjectStruct::GetSequences(char*** theSequences)
{
 *theSequences = new char*[numSequences] ;

 HvObjSequence* temp = theSequencesListHeader ;
 int total  = 0 ;

 while (temp != 0)
        {
                (*theSequences)[total] = new
char[strlen(temp->clipName)+strlen(temp->clipSequence)+3] ;

sprintf((*theSequences)[total++],"%s::%s",temp->clipName,temp->clipSequence
) ;
          temp = temp->next ;
        }

 return numSequences ;
}


void HvObjectStruct::DelSequence(const char* clip, const char* seq)
{
 HvObjSequence* temp = theSequencesListHeader ;
```

Copyright 1998 Veon, Ltd.

```
HvObjSequence* prevTemp = 0 ;

while (temp != 0)
        if ( (strcmp(temp->clipName, clip) == 0) &&
                        (strcmp(temp->clipSequence, seq) == 0) )   // found it
                {
                        if (prevTemp != 0)
                                prevTemp->next = temp->next ;
                        else
                                theSequencesListHeader = temp->next ;
                        numSequences-- ;
                        break ;
                }
        else
                {
                 prevTemp = temp ;
                 temp = temp->next ;
                }
}
//
// a method to write an object file
//
int HvObjectStruct::Write(const char *FilePath)
{
        ofstream outfile ;

        if (FilePath == 0)
          return HV_ERROR ;

        outfile.open(FilePath, ios::out ) ;

        Write(outfile) ;

        outfile.close() ;
```

Copyright 1998 Veon, Ltd.

```
        return HV_OK ;
}

//
// a debug method to dump to file
//
int HvObjectStruct::Dump(const char *FilePath)
{
        ofstream outfile ;

        if (FilePath == 0)
          return HV_ERROR ;

        outfile.open(FilePath, ios::out ) ;

        Dump(outfile) ;

        return HV_OK ;
}

//
// a method to write an object file
//
void HvObjectStruct::Write(ostream& os)
{
  os <<  id  << "\n" ;

  os << numSequences << "\n" ;
  for (HvObjSequence* temp = theSequencesListHeader ; temp != 0 ; temp=
temp->next)
         os << temp->clipName << " " << temp->clipSequence << "\n" ;

  if (theTarget != 0)
        os << strlen(theTarget) << " " << theTarget << "\n" ;
```

Copyright 1998 Veon, Ltd.

```
 else
  os << "0\n" ;
}

//
// a debug method to dump all info
//
void HvObjectStruct::Dump(ostream& os)
{
 os << " dumping an HvObjectStruct" << "\n" ;

 os <<  " object name: " << id  << "\n" ;

 os << " sequences defined: " << numSequences << "\n" ;
 for (HvObjSequence* temp = theSequencesListHeader ; temp != 0 ; temp=
temp->next)
        os << "from clip: " << temp->clipName << "  called: " <<
temp->clipSequence << "\n" ;

 if (theTarget != 0)
        os << "target:  " << theTarget << "\n" ;
 else
        os << " No target !! \n" ;

 os << " end of dump" << "\n" ;
}

#if !defined(__hvobjfile_h)
#define __hvobjfile_h
```

Copyright 1998 Veon, Ltd.

```
/*  HV Project
        Ephyx Technologies Ltd.
        Copyright © 1994. All Rights Reserved.

        SUBSYSTEM:    hvtest.exe Application
        FILE:         hvobj.h



        OVERVIEW
        ========
        Class definition for a clip file.
*/


#include <stdio.h>
#include <string.h>

#include <iostream.h>
#include <fstream.h>

#include "hvclp.h"
// general macros


class HvTarget ;

class HvObjSequence {
public:
  char *clipName ;
  char *clipSequence ;
  HvObjSequence* next ;
  HvObjSequence(const char* clip, const char* seq, HvObjSequence* n) ;
  char *Clip() { return clipName ; }
```

Copyright 1998 Veon, Ltd.

```
 char *Sequence() { return clipSequence ; }
} ;

class HvObjectStruct {
public:  // debug
 char* id ;                                          // identifier of the object
 char* theTarget ;              // target

//  HvObjSequence* theSequencesList ;
 HvObjSequence* theSequencesListHeader ;
 int numSequences ;

public:
 HvObjectStruct() {id = 0 ; theTarget = 0 ; }
        // empty ctor
 HvObjectStruct(const char* id) ;                    // full ctor
 HvObjectStruct(int &rc,const char* FilePath) ;      // full ctor from file
 virtual ~HvObjectStruct() ;                                 // dtor

 int AddObjSequence(HvObjSequence* seq) ;                 // adds a sequence to
the object's definition
 int GetSequences(HvObjSequence** theSequences) ;
 int GetSequences(char*** theSequences) ;
 int AddObjSequence(const char*clip, const char* seq) ;  // adds a sequence.
 int ExistObjSequence(const char*clip, const char* seq) ;  // checks if this
sequence exists.
 void DelSequence(const char*clip, const char* seq) ;  // deletes an existing
sequence.

 int Write(const char* FilePath) ;  // write to file
 void Write(ostream& os) ;                           // write to a stream

 int Dump(const char* FilePath) ;  // dump to file
 void Dump(ostream& os) ;                            // dump to a stream
```

Copyright 1998 Veon, Ltd.

```
  void SetTarget(const char* tg) { if (tg != 0) theTarget = strdup(tg) ; }
  char* Target() { return theTarget ; }
} ;

#endif
```

Copyright 1998 Veon, Ltd.

```
#include "stdafx.h"
#include "iostream.h"
#include "fstream.h"

#include "hvprj.h"
#include "hvflstr.h"

static char flstr[256] ;
#define M_FLSTR     256

int HvProject::GetClips(char ***Clips)
{
        // open the project file
        FILE *infile;
        int nc,no ;
        int numClips = 0 ;
        char line[256];

        infile = fopen(path,"rb");

        if (infile == 0)
          return 0 ;

        // read the first line:  <numClips>  <numObjects>
        EGetStr(infile,flstr,M_FLSTR) ;
        sscanf(flstr,"%d %d",&nc,&no);

        // allocate string array
        (*Clips) = new char* [20] ;
        if (*Clips == 0)
          return 0 ;

        while (EGetStr(infile,flstr,M_FLSTR))
        {
```

Copyright 1998 Veon, Ltd.

```
            sscanf(flstr,"%d %s",&no,line);
            if (no == HV_PRJ_CLIP)
                {
                    (*Clips)[numClips++] = strdup(line) ;
                }
        }

    // fill the rest with 0
    for (int i = nc ; i < 20 ; i++)
     (*Clips)[i] = 0 ;

    fclose(infile);

    return numClips ;
}

int HvProject::GetObjects(char ***Objects)
{
    // open the project file
    FILE *infile;
    int nc,no ;
    int numObjects = 0 ;
    char line[256];

    infile = fopen(path,"rb");

    if (infile == 0)
     return 0 ;

    // read the first line:  <numClips>  <numObjects>
    EGetStr(infile,flstr,M_FLSTR) ;
    sscanf(flstr,"%d %d",&nc,&no);

    // allocate string array
```

Copyright 1998 Veon, Ltd.

```
        (*Objects) = new char* [20] ;
        if (*Objects == 0)
         return 0 ;

        while (EGetStr(infile,flstr,M_FLSTR))
        {
                sscanf(flstr,"%d %s",&nc,line);
                if (nc == HV_PRJ_OBJECT)
                        {
                                (*Objects)[numObjects++] = strdup(line) ;
                        }
         }

        // fill the rest with 0
        for (int i = no ; i < 20 ; i++)
         (*Objects)[i] = 0 ;

        fclose(infile);

        return numObjects ;
}

// Store the project
int HvProject::Save(int numClips, char **Clips, int numObjects, char **Objects)
{
        ofstream outfile;

        outfile.open(path, ios::out) ;

        int nClips = 0 ;
        int nObjects = 0 ;

        // count actual number of clips
        for (int i = 0 ; i < numClips ; i++)
```

Copyright 1998 Veon, Ltd.

```
        if (Clips[i] != 0)
                nClips++ ;

        for (i = 0; i < numObjects ; i++)
        if (Objects[i] != 0)
                nObjects++ ;


        outfile << nClips << "  " << nObjects << "\n" ;

        for (i = 0 ; i < numClips ; i++)
        if (Clips[i] != 0)
                outfile << HV_PRJ_CLIP << " " << Clips[i] << "\n" ;

        for (i = 0 ; i < numObjects ; i++)
        if (Objects[i] != 0)
                outfile << HV_PRJ_OBJECT << " " << Objects[i] << "\n" ;

        outfile.close() ;

        return 1 ;
}

int HvProject::SaveAs( int numClips, char **Clips, int numObjects, char
**Objects, const char *path )
{
        delete this->path ;

        this->path = strdup(path) ;
        return Save(numClips, Clips, numObjects, Objects) ;
}


#if !defined(__hvprjfile_h)
```

Copyright 1998 Veon, Ltd.

```
#define __hvprjfile_h

/*  HV Project
        Ephyx Technologies Ltd.
        Copyright © 1994. All Rights Reserved.

        SUBSYSTEM:    hvtest.exe Application
        FILE:         hvprj.h



        OVERVIEW
        ========
        Class definition for a project file.
*/


#define HV_PRJ_CLIP 0
#define HV_PRJ_OBJECT 1

#include <string.h>

class HvProject
{
 public:

        char *path ; // the path of the project directory

 public:

        HvProject( char *p ) {path = strdup(p); }

        // Export the project data: clips, objects etc.
        int GetClips(char ***Clips) ;
```

Copyright 1998 Veon, Ltd.

```
        int GetObjects(char ***Objects) ;

        char* Path() { return path ; }

        // Store the project
        int Save(int numClips, char **Clips, int numObjects, char **Objects) ;
        int SaveAs( int numClips, char **Clips, int numObjects, char **Objects,
const char *path ) ;
};

#endif
```

Copyright 1998 Veon, Ltd.

```
// hvstrdoc.cpp : implementation of the CHvstreamDoc class
//

#include "stdafx.h"
#include "hvstream.h"

#include "hvstrdoc.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CHvstreamDoc

IMPLEMENT_DYNCREATE(CHvstreamDoc, CDocument)

BEGIN_MESSAGE_MAP(CHvstreamDoc, CDocument)
        //{{AFX_MSG_MAP(CHvstreamDoc)
                // NOTE - the ClassWizard will add and remove mapping macros
here.
                //    DO NOT EDIT what you see in these blocks of generated
code!
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CHvstreamDoc construction/destruction

CHvstreamDoc::CHvstreamDoc()
{
        // TODO: add one-time construction code here

}
```

Copyright 1998 Veon, Ltd.

```
CHvstreamDoc::~CHvstreamDoc()
{
}

BOOL CHvstreamDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;

        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CHvstreamDoc serialization

void CHvstreamDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
                // TODO: add storing code here
        }
        else
        {
                // TODO: add loading code here
        }
}



/////////////////////////////////////////////////////////////////////////////
```

Copyright 1998 Veon, Ltd.

```
// CHvstreamDoc diagnostics

#ifdef _DEBUG
void CHvstreamDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CHvstreamDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG
```

```
/////////////////////////////////////////////////////////////////////////////
// CHvstreamDoc commands
```

Copyright 1998 Veon, Ltd.

```
// hvstrdoc.h : interface of the CHvstreamDoc class
//
/////////////////////////////////////////////////////////////////////////////

class CHvstreamDoc : public CDocument
{
protected: // create from serialization only
        CHvstreamDoc();
        DECLARE_DYNCREATE(CHvstreamDoc)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CHvstreamDoc)
        public:
        virtual BOOL OnNewDocument();
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CHvstreamDoc();
        virtual void Serialize(CArchive& ar);   // overridden for document i/o
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:
```

Copyright 1998 Veon, Ltd.

```
// Generated message map functions
protected:
        //{{AFX_MSG(CHvstreamDoc)
                // NOTE - the ClassWizard will add and remove member functions
here.
                //    DO NOT EDIT what you see in these blocks of generated
code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

Copyright 1998 Veon, Ltd.

```
// hvstream.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "hvstream.h"

#include "mainfrm.h"
#include "hvstrdoc.h"
#include "hvstrvw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CHvstreamApp

BEGIN_MESSAGE_MAP(CHvstreamApp, CWinApp)
        //{{AFX_MSG_MAP(CHvstreamApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
                // NOTE - the ClassWizard will add and remove mapping macros
here.
                //    DO NOT EDIT what you see in these blocks of generated
code!
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CHvstreamApp construction

CHvstreamApp::CHvstreamApp()
```

Copyright 1998 Veon, Ltd.

```
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CHvstreamApp object

CHvstreamApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CHvstreamApp initialization

BOOL CHvstreamApp::InitInstance()
{
        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

        Enable3dControls();

        LoadStdProfileSettings();  // Load standard INI file options (including
MRU)

        // Register the application's document templates.  Document templates
        //  serve as the connection between documents, frame windows and views.

        CSingleDocTemplate* pDocTemplate;
        pDocTemplate = new CSingleDocTemplate(
                IDR_MAINFRAME,
                RUNTIME_CLASS(CHvstreamDoc),
                RUNTIME_CLASS(CMainFrame),      // main SDI frame
window
                RUNTIME_CLASS(CHvstreamView));
```

Copyright 1998 Veon, Ltd.

```
        AddDocTemplate(pDocTemplate);

        // create a new (empty) document
        OnFileNew();

        if (m_lpCmdLine[0] != '\0')
        {
                // TODO: add command line processing here
        }

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
        CAboutDlg();

// Dialog Data
        //{{AFX_DATA(CAboutDlg)
        enum { IDD = IDD_ABOUTBOX };
        //}}AFX_DATA

// Implementation
protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
        //{{AFX_MSG(CAboutDlg)
                // No message handlers
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
```

Copyright 1998 Veon, Ltd.

```
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CHvstreamApp::OnAppAbout()
{
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CHvstreamApp commands
```

Copyright 1998 Veon, Ltd.

```
// hvstream.h : main header file for the HVSTREAM application
//

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CHvstreamApp:
// See hvstream.cpp for the implementation of this class
//

class CHvstreamApp : public CWinApp
{
public:
	CHvstreamApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CHvstreamApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CHvstreamApp)
	afx_msg void OnAppAbout();
		// NOTE - the ClassWizard will add and remove member functions
here.
		//    DO NOT EDIT what you see in these blocks of generated
code !
	//}}AFX_MSG
```

Copyright 1998 Veon, Ltd.

```
    DECLARE_MESSAGE_MAP()
};
```

Copyright 1998 Veon, Ltd.

```
////////////////////////////////////////////////////////////////////////////
// hvstrvw.cpp : implementation of the CHvstreamView class
//

#include "stdafx.h"
#define _OLEAUTO_H_
#include <vfw.h>

#include "hvstream.h"

#include "hvstrdoc.h"
#include "hvstrvw.h"

#define EMPTY_CODE              "0"
#define ELLIPSE_CODE    "1"
#define RECT_CODE               "2"
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////////////////////////////////////////////
// CHvstreamView

IMPLEMENT_DYNCREATE(CHvstreamView, CView)

BEGIN_MESSAGE_MAP(CHvstreamView, CView)
        //{{AFX_MSG_MAP(CHvstreamView)
        ON_COMMAND(ID_STREAM_ADDSTREAM, OnStreamAddstream)
        ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
```

Copyright 1998 Veon, Ltd.

```
// CHvstreamView construction/destruction

CHvstreamView::CHvstreamView()
{
        // TODO: add construction code here

}

CHvstreamView::~CHvstreamView()
{
}

/////////////////////////////////////////////////////////////////////////////
// CHvstreamView drawing

void CHvstreamView::OnDraw(CDC* pDC)
{
        CHvstreamDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        // TODO: add draw code for native data here
}

/////////////////////////////////////////////////////////////////////////////
// CHvstreamView diagnostics

#ifdef _DEBUG
void CHvstreamView::AssertValid() const
{
        CView::AssertValid();
}

void CHvstreamView::Dump(CDumpContext& dc) const
{
```

Copyright 1998 Veon, Ltd.

```
        CView::Dump(dc);
}

CHvstreamDoc* CHvstreamView::GetDocument() // non-debug version is inline
{
ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CHvstreamDoc)));
        return (CHvstreamDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CHvstreamView message handlers
/*
void CHvstreamView::OnStreamAddstream()
{
        int i;
        AVISTREAMINFO strhdr, videohdr;
        PAVIFILE pfile = NULL;
        PAVISTREAM psText = NULL, psVideo = NULL;
        char szText[256];
        int iLen;
        HRESULT hr;
        DWORD dwTextFormat;
        char *szTitle; // name of the avi file
        char szMessage[256];
        DWORD numFrames;

        // get the file name
        char szFilter[] = "Video Files (*.avi) | *.avi | All Files (*.*) | *.* ||";
        CFileDialog dlg (TRUE, NULL, NULL, 0, szFilter, NULL);
        if (dlg.DoModal() != IDOK)
                return ;
```

Copyright 1998 Veon, Ltd.

```
CString fileName = dlg.GetPathName();
szTitle = strdup (dlg.m_ofn.lpstrFile);

AVIFileInit();

//
// Open the movie file for writing....
//

hr = AVIFileOpen(&pfile,              // returned file pointer
        (LPCSTR)szTitle,                    // file name
        OF_READWRITE,          // mode to open file with
        NULL);                 // use handler determined
                               // from file extension....
if (hr != AVIERR_OK)
        goto error;

// Get The video stream
hr = AVIFileGetStream (pfile, &psVideo, streamtypeVIDEO, 0);
if (hr != AVIERR_OK)
        goto error;

// get the stream info
hr = AVIStreamInfo (psVideo, &videohdr, sizeof(videohdr));

// Fill in the stream header for the text stream.

memset(&strhdr, 0, sizeof(strhdr));
strhdr.fccType            = streamtypeTEXT;
strhdr.fccHandler          = mmioFOURCC('D', 'R', 'A', 'W');
strhdr.dwScale            = videohdr.dwScale;
strhdr.dwRate             = videohdr.dwRate;
strhdr.dwSuggestedBufferSize  = sizeof(szText);
strhdr.rcFrame                                = videohdr.rcFrame;
```

Copyright 1998 Veon, Ltd.

```
//      SetRect (&strhdr.rcFrame, videohdr.rcFrame.left,
videohdr.rcFrame.bottom,
//                                      videohdr.rcFrame.right,
videohdr.rcFrame.bottom+30);
        SetRectEmpty(&strhdr.rcFrame);


        // ....and create the stream.
        hr = AVIFileCreateStream(pfile, &psText, (struct
_AVISTREAMINFOW*)&strhdr);
        if (hr != AVIERR_OK) {
                goto error;
        }


        dwTextFormat = sizeof(dwTextFormat);
        hr = AVIStreamSetFormat(psText, 0, &dwTextFormat,
sizeof(dwTextFormat));
        if (hr != AVIERR_OK) {
                goto error;
        }


        numFrames = videohdr.dwLength;


        for (i = 0; i < numFrames; i++) {
                iLen = wsprintf(szText, "%d %d %d %d %d", 0, 0, 100, 100, i);


//              DWORD dwFlags = (i%15 == 0) ? AVIIF_KEYFRAME : 0;
//              DWORD dwFlags = AVIIF_KEYFRAME;
                DWORD dwFlags = (AVIStreamIsKeyFrame(psVideo, i)) ?
AVIIF_KEYFRAME : 0;
                // ... and write it as well.
                        hr = AVIStreamWrite(psText,
                                i,
                                1,
                                szText,
                                iLen + 1,
                                dwFlags,
```

Copyright 1998 Veon, Ltd.

```
                    NULL,
                    NULL);
                if (hr != AVIERR_OK)
                    break;

        CDC *pDC = GetDC();
        wsprintf (szMessage, "frame # %d", i);
        pDC->DrawText (szMessage, -1, CRect (0, 0, 100, 100), DT_TOP |
DT_LEFT );
        ReleaseDC (pDC);
        }

error:
        //
        // Now close the file
        //
        if (psVideo)
                AVIStreamClose (psVideo);

        if (psText)
                AVIStreamClose(psText);

        if (pfile)
                AVIFileClose(pfile);

        AVIFileExit();

}
*/
void CHvstreamView::OnStreamAddstream()
{
        int i;
        AVISTREAMINFO strhdr, videohdr;
        PAVIFILE pfile = NULL;
```

Copyright 1998 Veon, Ltd.

```
PAVISTREAM psText = NULL, psVideo = NULL;
int iLen;
HRESULT hr;
DWORD dwTextFormat;
char szMessage[256];
DWORD numFrames;
char *szText;
AVIFileInit();

// iterate through all the clips in the project
    for (int clipNumber = 0; clipNumber < numClips; clipNumber++)
    {

        pfile = NULL;
        psText = psVideo = NULL;
        char fileName[256];
        strcpy (fileName, m_diskName);
        strcat (fileName, theClips[clipNumber]->mediaFile());
        // Open the movie file for reading and writing
        hr = AVIFileOpen(&pfile,              // returned file pointer
            (LPCSTR)fileName,                 // file name
            OF_READWRITE,          // mode to open file with
            NULL);                 // use handler determined
                                   // from file extension....
        if (hr != AVIERR_OK)
            goto error;

        // Get The video stream
        hr = AVIFileGetStream (pfile, &psVideo, streamtypeVIDEO, 0);
        if (hr != AVIERR_OK)
            goto error;

        // get the stream info for the video
        hr = AVIStreamInfo (psVideo, &videohdr, sizeof(videohdr));
```

Copyright 1998 Veon, Ltd.

```
// Fill in the stream header for the text stream.
memset(&strhdr, 0, sizeof(strhdr));
strhdr.fccType              = streamtypeTEXT;
strhdr.fccHandler           = mmioFOURCC('D', 'R', 'A', 'W');
strhdr.dwScale              = videohdr.dwScale; // 1;
strhdr.dwRate               = videohdr.dwRate;// /
videohdr.dwScale;
strhdr.dwSuggestedBufferSize  = 256; //sizeof(szText);
SetRectEmpty(&strhdr.rcFrame);


// create the text stream.
hr = AVIFileCreateStream(pfile, &psText, (struct
_AVISTREAMINFOW*)&strhdr);
if (hr != AVIERR_OK) {
        goto error;
}

dwTextFormat = sizeof(dwTextFormat);
hr = AVIStreamSetFormat(psText, 0, &dwTextFormat,
sizeof(dwTextFormat));
if (hr != AVIERR_OK) {
        goto error;
}

numFrames = videohdr.dwLength;

// loop the frames to write stream data
for (i = 0; i < numFrames; i++)
{
        HvClipStruct * theClip = theClips[clipNumber];
        HvClipROI* Rois = theClip->GetRoiOfFrame(i) ;
        CString text;
        if (Rois == 0)
                text = EMPTY_CODE;
        else
```

Copyright 1998 Veon, Ltd.

```
                    text = ELLIPSE_CODE ;
                HvObjectStruct *object;
                while (Rois != 0)
                {
                    CString roiText;
                    object = GetObjectFromTrack (theClip->name(),
theClip->GetSequence(Rois->id));
                    roiText.Format (" %d %d %d %d", Rois->x,
Rois->y, Rois->a, Rois->b);
                    roiText = roiText + " " + CString (object->id);
                    text +=roiText;
                    Rois = Rois->next ;
                }

                szText = strdup ((LPCTSTR)text);
                iLen = strlen (szText);

                DWORD dwFlags = (AVIStreamIsKeyFrame(psVideo, i))
? AVIIF_KEYFRAME : 0;
                // write the frame data.
                hr = AVIStreamWrite(psText, i, 1, szText,
        iLen, dwFlags, NULL, NULL);
                if (hr != AVIERR_OK)
                    break;

            CDC *pDC = GetDC();
            wsprintf (szMessage, "%s # %d", theClipsPaths[clipNumber], i);
            pDC->DrawText (szMessage, -1, CRect (0, 0, 300, 100),
DT_TOP | DT_LEFT );
            ReleaseDC (pDC);
            }

        error:
            //
```

Copyright 1998 Veon, Ltd.

```
        // Now close the file
        //
        if (psVideo)
                AVIStreamClose (psVideo);

        if (psText)
                AVIStreamClose(psText);

        if (pfile)
                AVIFileClose(pfile);

    }

    AVIFileExit();

    CDC *pDC = GetDC();
    wsprintf (szMessage, "Writing terminated");
    pDC->DrawText (szMessage, -1, CRect (0, 0, 300, 100), DT_TOP |
DT_LEFT );
    ReleaseDC (pDC);

}

void CHvstreamView::OnFileOpen()
{
    // get the file name
    char szFilter[] = "Obvious Project (*.obn) | *.obn | All Files (*.*) | *.* ||";
    CFileDialog dlg (TRUE, NULL, NULL, 0, szFilter, NULL);
    if (dlg.DoModal() != IDOK)
        return ;

    CString fileName = dlg.GetPathName();
    CString l_sCaption = "HV Stream Writer - " + fileName;
    SetWindowText (l_sCaption);
```

Copyright 1998 Veon, Ltd.

```
//allocate HvProject object
DeleteProject();

theProject = new HvProject ((char *)fileName.GetBuffer(256));
// set current directory to the correct machine
m_diskName[0] = fileName[0];
m_diskName[1] = ':';
m_diskName[2] = '\0';
int rc;
//retrieve the names of the clips in the file and create HvClip objects
numClips = theProject->GetClips(&theClipsPaths) ;
for (int i = 0  ; i < numClips ; i++)
 {
 // get the full clip name
 char * clipName = new char[256];
 strcpy (clipName, m_diskName);
 strcat (clipName, theClipsPaths[i]);
 theClips[i] = new HvClipStruct (rc, strdup(clipName) ) ;
 }
//retrieve the names of the objects in the file and create HvObject objects

numObjects = theProject->GetObjects(&theObjectsPaths) ;
for ( i = 0  ; i < numObjects ; i++)
 {
 // get the full object name
 char * objectName = new char[256];
 strcpy (objectName, m_diskName);
 strcat (objectName, theObjectsPaths[i]);
 theObjects[i] = new HvObjectStruct (rc, strdup(objectName) );

 }
```

Copyright 1998 Veon, Ltd.

```
}

void CHvstreamView::DeleteProject()
{
        // free all project data
}

HvObjectStruct* CHvstreamView::GetObjectFromTrack(char* clipName, char*
clipSequence)
{
 for (int i = 0 ; i < 20 ; i++)
        {
                if (theObjects[i] != 0)
                 {
                        if (theObjects[i]->ExistObjSequence(clipName,
clipSequence))
                                return theObjects[i] ;
                 }
        }

        return 0 ;
}
```

Copyright 1998 Veon, Ltd.

```
// hvstrvw.h : interface of the CHvstreamView class
//
/////////////////////////////////////////////////////////////////////////////

#include "hvclp.h"
#include "hvobj.h"
#include "hvprj.h"
#include "hvflstr.h"
class CHvstreamView : public CView
{
protected: // create from serialization only
        CHvstreamView();
        DECLARE_DYNCREATE(CHvstreamView)

// protected data members
        HvProject *theProject;

        HvClipStruct* theClips[20];
        char **theClipsPaths;
        int numClips;

        HvObjectStruct* theObjects[20];
        char **theObjectsPaths;
        int numObjects;

        char m_diskName[3];
// Attributes
public:
        CHvstreamDoc* GetDocument();

// Operations
public:

// Overrides
```

Copyright 1998 Veon, Ltd.

```
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CHvstreamView)
        public:
        virtual void OnDraw(CDC* pDC);  // overridden to draw this view
        protected:
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CHvstreamView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

        // free project allocated memory
        void DeleteProject();

        // get an object from a given clip and sequence
        HvObjectStruct* GetObjectFromTrack(char* clipName, char*
clipSequence);
// Generated message map functions
protected:
        //{{AFX_MSG(CHvstreamView)
        afx_msg void OnStreamAddstream();
        afx_msg void OnFileOpen();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in hvstrvw.cpp
inline CHvstreamDoc* CHvstreamView::GetDocument()
```

Copyright 1998 Veon, Ltd.

```
  { return (CHvstreamDoc*)m_pDocument; }
#endif
```

//

Copyright 1998 Veon, Ltd.

Copyright 1998 Veon, Ltd.

```
// mainfrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "hvstream.h"

#include "mainfrm.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
	//{{AFX_MSG_MAP(CMainFrame)
		// NOTE - the ClassWizard will add and remove mapping macros
here.
		//    DO NOT EDIT what you see in these blocks of generated
code !
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
	// TODO: add member initialization code here

}
```

Copyright 1998 Veon, Ltd.

```
CMainFrame::~CMainFrame()
{
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
}

#endif //_DEBUG
```

Copyright 1998 Veon, Ltd.

```
/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
```

Copyright 1998 Veon, Ltd.

```
// mainfrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
                // NOTE - the ClassWizard will add and remove member functions
here.
```

Copyright 1998 Veon, Ltd.

```
        //    DO NOT EDIT what you see in these blocks of generated
code!
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

Copyright 1998 Veon, Ltd.

```
//////////////////////////////////////////////////////////////////////////
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by hvstream.rc
//
#define IDD_ABOUTBOX                    100
#define IDR_MAINFRAME                   128
#define ID_STREAM_ADDSTREAM             32771

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                1
#define _APS_NEXT_RESOURCE_VALUE        130
#define _APS_NEXT_COMMAND_VALUE         32772
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

Copyright 1998 Veon, Ltd.

```
// stdafx.cpp : source file that includes just the standard includes

//      hvstream.pch will be the pre-compiled header

//      stdafx.obj will contain the pre-compiled type information



#include "stdafx.h"

/*  HV Project
        Ephyx Technologies Ltd.
        Copyright © 1994. All Rights Reserved.

        SUBSYSTEM:    hvtest.exe Application
        FILE:         hvclp.cpp



        OVERVIEW
        ========
        Class member methods for a clip file.


        file structure:
        <clip name>
        <video file full path>
        <video type>            // dropped for the time being
        <num markers>           each marker is <frame number> , <identifier>
                        <marker1>
                        ...
        <num sequences>
           <sequence>       each sequence is <seq number> , <identifier>
```

Copyright 1998 Veon, Ltd.

ROIS: for each frame describe the relevant ROIs.

<t> <id> <x> <y> <a> <b> <theta> (where id is of the sequence)

....

*/

```
#include "stdafx.h"
#include "hvclp.h"
#include "hvflstr.h"

#include "iostream.h"
#include "fstream.h"

static char flstr[255];
//
// definition ctor
//
HvClipStruct::HvClipStruct(int &rc, const char *name, HvVideoTypes type) :
_type(type)
{
 rc = HV_OK ;

 maxFramesForRoi = _numMarkers = 0 ;

 // start with an allocation of markers
 _numAllocatedMarkers = 4 ;

 _listOfMarkers = 0 ;
 _listOfRoiByFrame = 0 ;

 _listOfMarkers = new HvClipMarker [_numAllocatedMarkers] ;
 if (_listOfMarkers == 0)
        {
                rc = HV_ERROR ;
```

Copyright 1998 Veon, Ltd.

```
                return ;
        }

_numAllocatedSequences = 4 ;
_numSequences = 0 ;
_bagOfSequences = new HvSequence [ _numAllocatedSequences ] ;
if (_bagOfSequences == 0)
        {
                rc = HV_ERROR ;
                return ;
        }

_name = 0 ;

// copy the given name
if (name != 0)
        {
                _name = new char [strlen(name)+1] ;
                if (_name != 0 )
                 strcpy(_name, name) ;
        }

_mediaFilePath = 0 ;

}

//
// a ctor from a file path:
//  opens the file and reads the info
//  rc = -1 if fails
HvClipStruct::HvClipStruct( int &rc, const char *filePath ) : _type(HV_AVI)
{
        FILE *fileDesc ;
        char line[256] ;
```

Copyright 1998 Veon, Ltd.

```
rc = HV_OK ;

if (filePath != 0 && filePath[0] != '\0')
        {
         // open the file for read only
         if ((fileDesc = fopen(filePath, "rb")) == 0 )
                {
                        rc = HV_ERROR ;
                        return ;
                }

         // get the name of the clip
         if (!EGetStr(fileDesc,flstr,255))
                {
                        rc = HV_ERROR ;
                        fclose(fileDesc) ;
                        return ;
                }
         sscanf(flstr,"%s",line) ;
         // allocate and copy the data
         if ((_name  = new char[strlen(line) + 1]) == 0)
                {
                        rc = HV_ERROR ;
                        fclose(fileDesc) ;
                        return ;
                }
         strcpy(_name, line) ;

         // get the full path of the media file
         if (!EGetStr(fileDesc,flstr,255))
                {
                        rc = HV_ERROR ;
                        fclose(fileDesc) ;
                        return ;
```

Copyright 1998 Veon, Ltd.

```
        }
    sscanf(flstr,"%s",line) ;
    // allocate and copy the data
    if ((_mediaFilePath  = new char[strlen(line) + 1]) == 0)
        {
            rc = HV_ERROR ;
            fclose(fileDesc) ;
            return ;
        }
    strcpy(_mediaFilePath, line) ;

    // get the video type
    // the type is assumed AVI
    _type = HV_AVI ;

    // read the markers set for the clip
    if (!EGetStr(fileDesc,flstr,255))
        {
            rc = HV_ERROR ;
            fclose(fileDesc) ;
            return ;
        }
    sscanf(flstr,"%d",&_numMarkers) ;
    // allocate and copy the data ( allocate a bit more in case we'll
need to add later
    if (_numMarkers > 0)
        {
            _listOfMarkers = new HvClipMarker
[_numMarkers+3] ;
            _numAllocatedMarkers = _numMarkers + 3 ;

            if (_listOfMarkers == 0)
             {
                rc = HV_ERROR ;
                fclose(fileDesc) ;
```

Copyright 1998 Veon, Ltd.

```
                    return ;
                }
            }
        else
            {
                _numAllocatedMarkers = 0 ;
                _listOfMarkers = 0 ;
            }

        // loop and read all the markers
        for ( int i = 0 ; i < _numMarkers ; i++)
            {
                if (!EGetStr(fileDesc,flstr,255))
                    {
                        rc = HV_ERROR ;
                        fclose(fileDesc) ;
                        return ;
                    }
                sscanf(flstr, "%ld %s", &_listOfMarkers[i].mark,
line) ;
                _listOfMarkers[i].id = strdup(line) ;
            }

        // read the sequences for the clip
         if (!EGetStr(fileDesc,flstr,255))
            {
                rc = HV_ERROR ;
                fclose(fileDesc) ;
                return ;
            }
         sscanf(flstr,"%d",&_numSequences) ;
                // allocate and copy the data ( allocate a bit more in case
we'll need to add later
         if (_numSequences > 0)
            {
```

Copyright 1998 Veon, Ltd.

```
                    _bagOfSequences = new HvSequence
[_numSequences +3] ;
                    _numAllocatedSequences = _numSequences +3 ;

                    if (_bagOfSequences == 0)
                     {
                          rc = HV_ERROR ;
                          fclose(fileDesc) ;
                          return ;
                     }
               }
          else
               {
                _numAllocatedSequences = 0 ;
                    _bagOfSequences = 0 ;
               }

          // loop and read all the sequences
          for ( i = 0 ; i < _numSequences ; i++)
               {
                    if (!EGetStr(fileDesc,flstr,255))
                         {
                              rc = HV_ERROR ;
                              fclose(fileDesc) ;
                              return ;
                         }
                    sscanf(flstr, "%ld %s", &_bagOfSequences[i].id,
line) ;
                    _bagOfSequences[i].name = strdup(line) ;
               }

          // loop and read all ROIs
          maxFramesForRoi = 0 ;
          {
               int t,a,b,x,y;
```

Copyright 1998 Veon, Ltd.

```
long hnd;
float th ;

while (EGetStr(fileDesc,flstr,255))
    {
sscanf(flstr,"%d %ld %d %d %d %d
%f",&t,&hnd,&x,&y,&a,&b,&th) ;
if (t >= maxFramesForRoi)
    // allocate another chunk of frame
slots
    if (!allocFrameSlots(t))
     {
        rc = HV_ERROR ;
        fclose(fileDesc) ;
        return ;
     }

// put the data in the right slot
if (_listOfRoiByFrame[t] != 0)
 {
     HvClipROI* temp = new
HvClipROI(hnd, x, y, a, b, th,

_listOfRoiByFrame[t],0 ) ;
     _listOfRoiByFrame[t]->prev =
temp ;
     _listOfRoiByFrame[t] = temp ;
 }
else
 {
     _listOfRoiByFrame[t] = new
HvClipROI(hnd, x, y, a, b, th,

_listOfRoiByFrame[t],0 ) ;
 }
// connect it to the sequence list
```

Copyright 1998 Veon, Ltd.

```
                                        AddToSequence(_listOfRoiByFrame[t]) ;
                                }
                        }

                        fclose(fileDesc) ;
                }
        else
                {
                        rc = HV_ERROR ;
                }

        return ;
}

//
// a virtual dtor
//
HvClipStruct::~HvClipStruct()
{
        HvClipROI* temp ;

        if (_name != 0)
          delete _name ;

        if (_mediaFilePath != 0)
          delete _mediaFilePath ;

        if (_numMarkers > 0)
          delete _listOfMarkers ;

        if (maxFramesForRoi > 0)
          for (int i = 0 ; i < maxFramesForRoi ; i++)
                if (( temp = _listOfRoiByFrame[i]) != 0 )
                  while (temp)
```

Copyright 1998 Veon, Ltd.

```
                {
                        HvClipROI *cur = temp ;
                        temp = temp->next ;
                        delete cur ;
                }
}

void HvClipStruct::setName(const char *name)
{
 if (name == 0)
        return ;

 if (_name)
        delete _name ;

 _name = new char [ strlen(name) + 1 ] ;
 if (_name != 0 )
        strcpy(_name , name ) ;

 return ;
}

void HvClipStruct::setMediaFile(const char *name)
{
 if (name == 0)
        return ;

 if (_mediaFilePath)
        delete _mediaFilePath ;

 _mediaFilePath = new char [ strlen(name) + 1 ] ;
 if (_mediaFilePath != 0 )
        strcpy(_mediaFilePath , name ) ;
```

Copyright 1998 Veon, Ltd.

```
 return ;
}

//
// AddToSequence:   adds a ROI to a sequences (always at the end)
//
void HvClipStruct::AddToSequence(HvClipROI *roi)
{
   int set = -1 ;
        // find the right sequence
        for (int i = 0 ; i < _numSequences ; i++)
                if (_bagOfSequences[i].id == roi->id)
                        set = i ;

        if (set < 0)
                return ;        // no such sequence defined (must define a sequence
first )

        // assume it is a new LAST roi
        if (_bagOfSequences[set].first == 0)  // no rois until now
                _bagOfSequences[set].first = _bagOfSequences[set].last = roi ;
        else
                {
                _bagOfSequences[set].last->nextInSequence = roi ;
                _bagOfSequences[set].last = roi ;
                }
}

//
//  GetSequence : retrieves a sequence name
//
//
char* HvClipStruct::GetSequence( long id )
{
```

Copyright 1998 Veon, Ltd.

```
        if (id < 0)
         return 0 ;

        for (int i = 0 ; i < _numAllocatedSequences ; i++)
        {
                if (_bagOfSequences[i].id == id)
                 {
                        return _bagOfSequences[i].name ;
                 }
        }

        return 0 ;
}

//
// DelSequence : deletes a sequence to the clip's list
//
//        return HV_OK if all OK, otherwise HV_ERROR
//
//
int HvClipStruct::DelSequence( const char* name )
{
        if (name == 0)
         return HV_ERROR ;

        for (int i = 0 ; i < _numAllocatedSequences ; i++)
         {
                if (strcmp(_bagOfSequences[i].name,name) == 0)
                        return DelSequence(_bagOfSequences[i].id) ;
         }
        return HV_ERROR ;
}

//
```

Copyright 1998 Veon, Ltd.

```
// DelSequence : deletes a sequence to the clip's list
//
//        return HV_OK if all OK, otherwise HV_ERROR
//
int HvClipStruct::DelSequence( long id )
{
        if (id < 0)
         return HV_ERROR ;

        for (int i = 0 ; i < _numAllocatedSequences ; i++)
         {
                if (_bagOfSequences[i].id == id)
                 {
                        // loop to free the ROIs
                        HvClipROI* current = _bagOfSequences[i].first ;
                        HvClipROI* temp ;
                        while (current != 0)
                                {
                                        if (current->prev != 0)  // BUG: we do not
how to clear first elements !!!
                                         {
                                                current->prev->next =
current->next ;
                                                if (current->next != 0)
                                                        current->next->prev =
current->prev ;

                                                temp = current ;
                                                current = current->nextInSequence
;
                                                delete temp ;
                                         }
                                        else            // all we do is skip
                                         {
                                                current = current->nextInSequence
;
```

Copyright 1998 Veon, Ltd.

```
                    }
                }

                _bagOfSequences[i].id = 0L ;

                _numSequences-- ;
                if (_numSequences < 0) _numSequences = 0 ;

                return HV_OK ;
            }
        }

        return HV_ERROR ;
}

long HvClipStruct::GetSequenceId()
{
 // start with numsequences+1 and make sure
 long id = _numSequences+1 ;

 while (IdExists(id)) id++ ;

 return id ;
}

int HvClipStruct::IdExists(const int id)
{
        for (int i = 0; i < _numAllocatedSequences; i++)
            if (_bagOfSequences[i].id == id)
             return 1 ;

        return 0 ;
}
```

Copyright 1998 Veon, Ltd.

```
//
//  AddClipSequence : adds a sequence to the clip's list
//
//          return HV_OK if all OK, otherwise HV_ERROR
//
int HvClipStruct::AddClipSequence( const char *name )
{
        if ( name == 0)
          return HV_ERROR ;

        if (_numSequences >= _numAllocatedSequences)
                // allocate another chunk of frame slots
                if (!allocSequenceSlots())
                 return HV_ERROR ;

        long id  = GetSequenceId() ;

        // find an empty spot
        int set  = 0 ;
        while (_bagOfSequences[set].id != 0) set++ ;

        // put the data in the right slot
        _bagOfSequences[set].id = id ;
        _bagOfSequences[set].name = strdup(name) ;

        _numSequences++ ;

        return id ;
}

//
//  GetSequences : returns the number and names of all sequences defined
//
int HvClipStruct::GetSequences(char *** names)
```

Copyright 1998 Veon, Ltd.

```
{
        int total = 0;

        if (_numSequences == 0)
         return 0 ;

        (*names) = new char* [_numSequences] ;
        if ((*names) == 0)
         return HV_ERROR ;

        for (int i = 0 ; i < _numAllocatedSequences; i++)
         {
                if (_bagOfSequences[i].id != 0L)  //  then it is a sequence
                        (*names)[total++] = strdup(_bagOfSequences[i].name) ;
         }

        return total ;
}

//
//  GetSequencesExt : returns the number and names of all sequences defined
//                                                          concatenating the clip's
name at the beginning of each sequence.
//
int HvClipStruct::GetSequencesExt(char *** names)
{
        int total = 0;

        if (_numSequences == 0)
         return 0 ;

        (*names) = new char* [_numSequences] ;
        if ((*names) == 0)
         return HV_ERROR ;
```

Copyright 1998 Veon, Ltd.

```
        for (int i = 0 ; i < _numAllocatedSequences; i++)
         {
                if (_bagOfSequences[i].id != 0L)  //  then it is a sequence
                        {
                         (*names)[total] = new
char[strlen(_name)+strlen(_bagOfSequences[i].name)+3] ;
                         sprintf((*names)[total++],"%s::%s",_name,
_bagOfSequences[i].name) ;
                        }
         }

        return total ;
}


//
//  addMarker : adds a marker to the clip's list
//
//         return HV_OK if all OK, otherwise HV_ERROR
//
int HvClipStruct::addMarker( long t,char *id )
{
        if (t < 0 || id == 0)
         return HV_ERROR ;

        if (_numMarkers >= _numAllocatedMarkers)
                // allocate another chunk of frame slots
                if (!allocMarkerSlots())
                 return HV_ERROR ;

        // put the data in the right slot
        _listOfMarkers[_numMarkers].mark = t ;
        _listOfMarkers[_numMarkers].id = strdup(id) ;
```

Copyright 1998 Veon, Ltd.

```
        _numMarkers++ ;

        return HV_OK ;
}

//
// addRoi : adds a ROI element to the clip's list
//
//        return HV_OK if all OK, otherwise HV_ERROR
//
int HvClipStruct::addRoi( long t,long handle, int x, int y, int a, int b, float theta )
{
        if (t >= maxFramesForRoi)
                // allocate another chunk of frame slots
                if (!allocFrameSlots(t))
                 return HV_ERROR ;

        // put the data in the right slot
        if (_listOfRoiByFrame[(int)t] == 0)
          _listOfRoiByFrame[(int)t] = new HvClipROI(handle, x, y, a, b, theta, 0,
0) ;
        else
         {
                HvClipROI* temp = new HvClipROI(handle, x, y, a, b, theta,

                                        _listOfRoiByFrame[(int)t],0 ) ;
                _listOfRoiByFrame[(int)t]->prev = temp ;
                _listOfRoiByFrame[(int)t] = temp ;
         }

        // connect it to the sequence list
        AddToSequence(_listOfRoiByFrame[t]) ;

        return HV_OK ;
}
```

Copyright 1998 Veon, Ltd.

```
long HvClipStruct::ObjectAtFramePoint(const long t, const int x, const int y)
const
{
        long hnd ;

        if (t < 0 )
         return -1L ;

        if (t >= maxFramesForRoi)
         return -1L ;
        HvClipROI* temp = _listOfRoiByFrame[(int)t] ;

        // go thru the list of frames at time t
        while (temp != 0)
         {
                if (( hnd = temp->PtInROI(x,y) ) != -1L)
                        return hnd ;
                temp = temp->next ;
         }

        // did not find anything
        return -1L ;
}

//
// allocFrameSlots :: increase memory allocation for ROIs
//
int HvClipStruct::allocFrameSlots(const long minFrame)
{
        int upTo = (( (int)minFrame >= ( 2 * maxFramesForRoi )) ?
(int)minFrame + 1 : 2 * maxFramesForRoi ) ;

        // alloc at least 10
        upTo = ( upTo < 10 ) ? 10 : upTo ;
```

Copyright 1998 Veon, Ltd.

```
        // alloc a new array
        HvClipROI** temp = new HvClipROI* [upTo] ;

        // alloc problems
        if (temp == 0 )
         return 0 ;

        // clear the array
        for (int i = 0 ; i < upTo ; i++)
         temp[i] = 0 ;

        if (_listOfRoiByFrame != 0)
         {
                // copy what we already have
                for (i = 0 ; i < maxFramesForRoi ; i++)
                 temp[i] = _listOfRoiByFrame[i] ;

         }

        // set the new max number
        maxFramesForRoi = upTo ;

        _listOfRoiByFrame = temp ;

        return 1 ;
}

//
// allocMarkerSlots :: increase memory allocation for markers
//
int HvClipStruct::allocMarkerSlots()
{
        HvClipMarker* temp = new HvClipMarker[_numAllocatedMarkers + 10
] ;
```

Copyright 1998 Veon, Ltd.

```
        if (temp == 0)
         return 0 ;

        _numAllocatedMarkers += 10 ;

        // copy the old markers
        for (int i = 0 ; i < _numMarkers ; i++)
         temp[i] = _listOfMarkers[i] ;

        delete _listOfMarkers ;

        _listOfMarkers = temp ;

        return 1 ;
}

//
// allocSequenceSlots :: increase memory allocation for sequences
//
int HvClipStruct::allocSequenceSlots()
{
        HvSequence* temp = new HvSequence[_numAllocatedSequences + 10 ] ;

        if (temp == 0)
         return 0 ;

        _numAllocatedSequences += 10 ;

        // copy the old markers
        for (int i = 0 ; i < _numSequences ; i++)
         temp[i] = _bagOfSequences[i] ;

        delete _bagOfSequences ;
```

Copyright 1998 Veon, Ltd.

```
        _bagOfSequences = temp ;

        return 1 ;
}

//
// a method to write a clip file
//
int HvClipStruct::Write(const char *filePath)
{
        ofstream outfile ;

        if (filePath == 0)
          return HV_ERROR ;

        outfile.open(filePath, ios::out ) ;

        Write(outfile) ;

        outfile.close() ;

        return HV_OK ;
}

//
// a debug method to dump to file
//
int HvClipStruct::Dump(const char *filePath)
{
        ofstream outfile ;

        if (filePath == 0)
          return HV_ERROR ;
```

Copyright 1998 Veon, Ltd.

```
        outfile.open(filePath, ios::out ) ;

        Dump(outfile) ;

        return HV_OK ;
}

//
// a mathod to write a clip file
//
void HvClipStruct::Write(ostream& os)
{
 os << _name  << "\n" ;
 os << _mediaFilePath  << "\n" ;
//  os << " video type:  AVI \n" ;

 os << _numMarkers << "\n" ;
 for (int i =0 ; i< _numMarkers ; i++)
        os << _listOfMarkers[i].mark << " " << _listOfMarkers[i].id << "\n" ;

 os << _numSequences << "\n" ;
 for (i =0 ; i< _numAllocatedSequences ; i++)
        if (_bagOfSequences[i].id != 0)
          os << _bagOfSequences[i].id << " " << _bagOfSequences[i].name <<
"\n" ;

 for (i = 0 ; i < maxFramesForRoi ; i++)
        {
                HvClipROI* temp = _listOfRoiByFrame[i] ;
                if ( temp != 0 )
                        while (temp )
                         {
                                os << i << " " << temp->id << " " ;
                                os << temp->x << " " << temp->y << " " ;
```

Copyright 1998 Veon, Ltd.

```
                        os << temp->a << " " << temp->b << " " ;
                        os << temp->theta << "\n" ;
                        temp = temp->next ;
                    }
            }
}

//
// a debug mathod to dump all info
//
void HvClipStruct::Dump(ostream& os)
{
 int i;

 os << " dumping an HvClipStruct" << "\n" ;

 os << " clip name: " << _name  << "\n" ;
 os << " video file path: " << _mediaFilePath  << "\n" ;
 os << " video type:  AVI \n" ;
 os << " markers defined: " << _numMarkers << "\n" ;
 for ( i = 0 ; i< _numMarkers ; i++)
        os << "    \"" << _listOfMarkers[i].id << "\"  at " <<
_listOfMarkers[i].mark << "\n" ;

 os << " sequences defined: " << _numSequences << "\n" ;
 for ( i = 0 ; i< _numAllocatedSequences ; i++)
        if (_bagOfSequences[i].id != 0)
         os << "    \"" << _bagOfSequences[i].name << " referenced: " <<
_bagOfSequences[i].id << "\n" ;

 os << " ROIs defined: \n" ;
 for (i = 0 ; i < maxFramesForRoi ; i++)
        {
                HvClipROI* temp = _listOfRoiByFrame[i] ;
                if ( temp != 0 )
```

Copyright 1998 Veon, Ltd.

```
                while (temp )
                 {
                        os << "   object: " << temp->id << "\n" ;
                        os << "     in frame   " << i << "\n" ;
                        os << "     placed at  " << temp->x << "," <<
temp->y << "\n" ;
                        os << "     parameters " << temp->a << "," <<
temp->b <<
                       "," <<  temp->theta << "\n" ;
                        temp = temp->next ;
                 }
        }

 os << " end of dump" << "\n" ;
}



//
//  Class HvSequence
//
HvSequence::HvSequence(long id, char *name,HvClipROI* first, HvClipROI*
last)
{
 this->id = id ;
 if (name !=0)
        this->name = strdup(name);
 this->first = first ;
 this->last = last ;
}

HvSequenceIterator::HvSequenceIterator(HvSequence* sq)
{
 seq = sq ;
```

Copyright 1998 Veon, Ltd.

```
 current = sq->first ;
}

HvClipROI* HvSequenceIterator::Next()
{
 if (current != 0)
        {
                current = current->nextInSequence ;
        }
 return  current ;
}

HvClipROI* HvSequenceIterator::First()
{
 if (current != 0)
        {
                if (seq != 0)
                        current = seq->first ;
        }
 return  current ;
}

HvClipROI* HvSequenceIterator::Last()
{
 if (current != 0)
        {
                if (seq != 0)
                        current = seq->last ;
        }
 return  current ;
}

long HvClipROI::PtInROI(const int x , const int y ) const
{
```

Copyright 1998 Veon, Ltd.

```
        float x1 = (float)(x - this->x) ;
        float y1 = (float)(y - this->y) ;
        if ( ((x1 / a) * (x1 / a) + (y1 / b) * (y1 / b)) > 1.0)
          return -1L ;

        return id ;
}
```

==========

MICROSOFT FOUNDATION CLASS LIBRARY : HVSTREAM

==========

AppWizard has created this HVSTREAM application for you. This application not only demonstrates the basics of using the Microsoft Foundation classes but is also a starting point for writing your application.

This file contains a summary of what you will find in each of the files that make up your HVSTREAM application.

Copyright 1998 Veon, Ltd.

HVSTREAM.MAK

This project file is compatible with the Visual C++ development environment.

It is also compatible with the NMAKE program provided with Visual C++.

To build a debug version of the program from the MS-DOS prompt, type
nmake /f HVSTREAM.MAK CFG="Win32 Debug"
or to build a release version of the program, type
nmake /f HVSTREAM.MAK CFG="Win32 Release"

HVSTREAM.H

This is the main header file for the application. It includes other project specific headers (including RESOURCE.H) and declares the CHvstreamApp application class.

HVSTREAM.CPP

This is the main application source file that contains the application class CHvstreamApp.

HVSTREAM.RC

This is a listing of all of the Microsoft Windows resources that the program uses. It includes the icons, bitmaps, and cursors that are stored in the RES subdirectory. This file can be directly edited in the Visual C++ development environment.

RES\HVSTREAM.ICO

This is an icon file, which is used as the application's icon. This icon is included by the main resource file HVSTREAM.RC.

RES\HVSTREAM.RC2

This file contains resources that are not edited by the Visual C++ development environment. You should place all resources not

Copyright 1998 Veon, Ltd.

editable by the resource editor in this file.

HVSTREAM.CLW

This file contains information used by ClassWizard to edit existing classes or add new classes. ClassWizard also uses this file to store information needed to create and edit message maps and dialog data maps and to create prototype member functions.

//

For the main frame window:

mainfrm.H, mainfrm.CPP

These files contain the frame class CMainFrame, which is derived from CFrameWnd and controls all SDI frame features.

//

AppWizard creates one document type and one view:

hvstrdoc.H, hvstrdoc.CPP - the document

These files contain your CHvstreamDoc class. Edit these files to add your special document data and to implement file saving and loading (via CHvstreamDoc::Serialize).

hvstrvw.H, hvstrvw.CPP - the view of the document

These files contain your CHvstreamView class.
CHvstreamView objects are used to view CHvstreamDoc objects.

Copyright 1998 Veon, Ltd.

//

Other standard files:

STDAFX.H, STDAFX.CPP

These files are used to build a precompiled header (PCH) file named HVSTREAM.PCH and a precompiled types file named STDAFX.OBJ.

RESOURCE.H

This is the standard header file, which defines new resource IDs. Visual C++ reads and updates this file.

//

Other notes:

AppWizard uses "TODO:" to indicate parts of the source code you should add to or customize.

//

Copyright 1998 Veon, Ltd.

We claim:

1. A data stream, comprising a hypervideo data stream including hypervideo data associated with an instance in time in a video.

2. The data stream of claim 1, wherein the hypervideo data stream further comprises target data associated with the instance in time.

3. The data stream of claim 1, wherein the hypervideo data stream further comprises geometry data of a hotspot associated with the instance in time.

4. The data stream of claim 3, wherein the hypervideo data stream further comprises shape data of a hotspot associated with the instance of time.

5. The data stream of claim 1, wherein the instance in time corresponds to a frame of the video.

6. A multimedia stream, comprising:

a hypervideo data stream including,

a header, including static hypervideo data;

a body, coupled to the header, including dynamic hypervideo data; and

wherein the dynamic hypervideo data includes time data that is associated with an instance of time in a video with which the dynamic hypervideo data and the static hypervideo data are associated.

7. The multimedia stream of claim 6, wherein the static hypervideo data comprises target data.

8. The multimedia stream of claim 6, wherein the dynamic hypervideo data further comprises geometry data of a hotspot.

9. The multimedia stream of claim 6, wherein the dynamic hypervideo data further comprises shape data of a hotspot.

10. The multimedia stream of claim 6, further comprising a video stream including a video frame, wherein the instance in time corresponds to the video frame.

11. The multimedia stream of claim 6, further comprising an audio stream.

12. A method of creating a multimedia stream, comprising:

authoring a hypervideo, including a video; and

exporting hypervideo parameters into a hypervideo data stream of a media file.

13. The method of claim 11, further comprising exporting a data stream of the video into the media file.

14. A method for authoring a hypervideo that is dynamic, comprising:

authoring a hypervideo; and

defining a hypervideo parameter that may vary during the hypervideo performance.

15. The method of claim 14, further comprising defining an event handler that issues a command to vary the parameter upon receiving an event message from the hypervideo.

Copyright 1998 Veon, Ltd.

16. The method of claim 15, wherein the defining of the event handler comprises defining an event handler defined by a logical condition including a variable defined by a query to a database.

17. The method of claim 14, wherein authoring the hypervideo comprises defining a hotspot and linking the hotspot to a target.

18. The method of performing a hypervideo that is dynamic, comprising:

performing a hypervideo; and

varying a parameter of the hypervideo during the performance of the hypervideo.

19. The method of claim 18, wherein the varying the parameter comprises varying the parameter upon the occurrence of an event in the hypervideo.

20. The method of claim 19, wherein the varying the parameter upon the occurrence of an event in the hypervideo comprises varying the parameter upon an event handler receiving a message associated with the event, and issuing a command to vary the parameter.

21. The method of claim 20, wherein the varying the parameter upon an event handler receiving a message associated with the event, and issuing a command to vary the parameter comprises varying the parameter upon the event handler evaluating a logical condition including a variable defined by a query to a database.

22. The method of claim 18, wherein the varying the parameter comprises varying the parameter that identifies a target.

23. A hypervideo system, comprising:

Copyright 1998 Veon, Ltd.

a dynamic hypervideo server;

a player coupled to the dynamic hypervideo server; and

a media server coupled to the player.

24. The hypervideo system of claim 23, further comprising a database.

25. The hypervideo system of claim 23, further comprising an application program.

26. The hypervideo system of claim 23, wherein the application program is an electronic commerce server.

27. The hypervideo system of claim 23, wherein the application program is an advertising server.

28. The hypervideo system of claim 23, wherein the application program is a training server.

29. The hypervideo system of claim 23, wherein the media server is a video server.

30. The hypervideo system of claim 23, further comprising a network coupling the player to the dynamic hypervideo server and the media server.

Copyright 1998 Veon, Ltd.

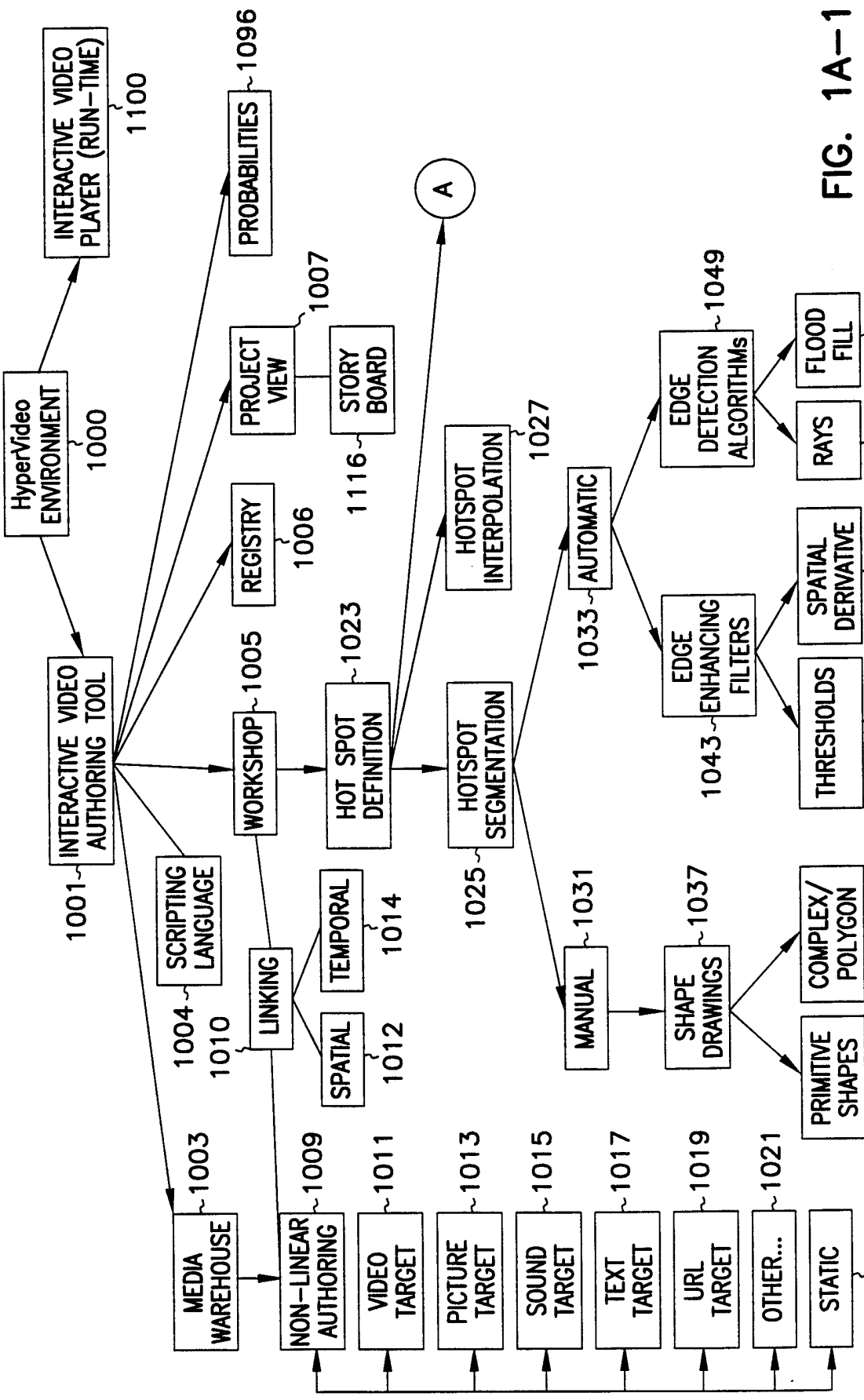

INTERACTIVE VIDEO PLAYER (RUN-TIME)
1100
HyperVideo ENVIRONMENT
1000
INTERACTIVE VIDEO AUTHORING TOOL
1001
PROBABILITIES
1096
PROJECT VIEW
1007
STORY BOARD
1116
REGISTRY
1006
WORKSHOP
1005
HOT SPOT DEFINITION
1023
A
HOTSPOT INTERPOLATION
1027
HOTSPOT SEGMENTATION
1025
AUTOMATIC
1033
MANUAL
1031
EDGE DETECTION ALGORITHMs
1049
FLOOD FILL
1053
RAYS
1051
EDGE ENHANCING FILTERS
1043
SPATIAL DERIVATIVE
1047
THRESHOLDS
1045
SHAPE DRAWINGS
1037
COMPLEX/ POLYGON
1041
PRIMITIVE SHAPES
1039
SCRIPTING LANGUAGE
1004
LINKING
1010
TEMPORAL
1014
SPATIAL
1012
MEDIA WAREHOUSE
1003
NON-LINEAR AUTHORING
1009
VIDEO TARGET
1011
PICTURE TARGET
1013
SOUND TARGET
1015
TEXT TARGET
1017
URL TARGET
1019
OTHER...
1021
STATIC
1022

FIG. 1A-1

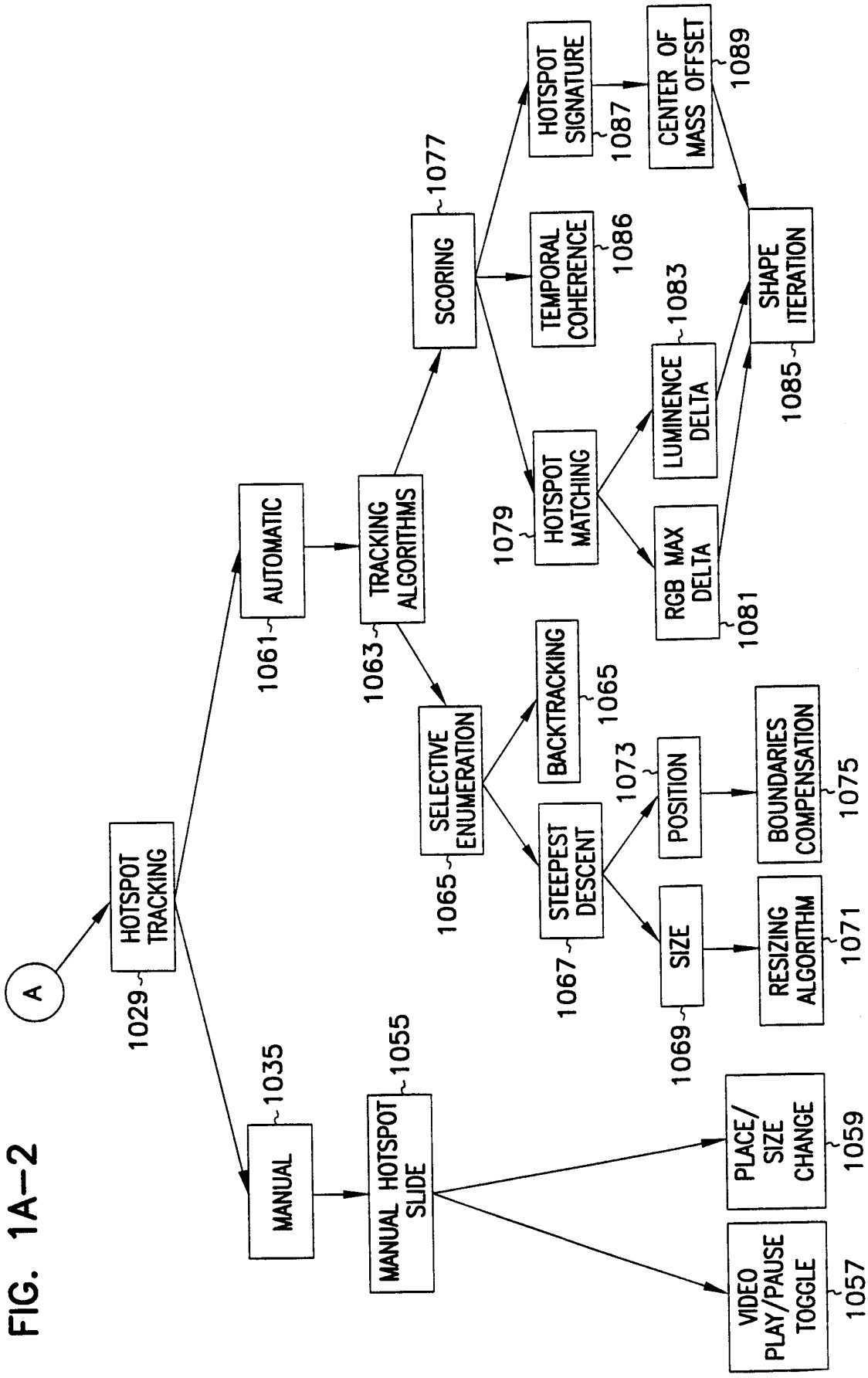

A
HOTSPOT TRACKING
1029
AUTOMATIC
1061
TRACKING ALGORITHMS
1063
SCORING
1077
HOTSPOT SIGNATURE
1087
CENTER OF MASS OFFSET
1089
TEMPORAL COHERENCE
1086
HOTSPOT MATCHING
1079
LUMINENCE DELTA
1083
RGB MAX DELTA
1081
SHAPE ITERATION
1085
SELECTIVE ENUMERATION
1065
BACKTRACKING
1065
STEEPEST DESCENT
1067
POSITION
1073
SIZE
1069
BOUNDARIES COMPENSATION
1075
RESIZING ALGORITHM
1071
MANUAL
1035
MANUAL HOTSPOT SLIDE
1055
PLACE/ SIZE CHANGE
1059
VIDEO PLAY/PAUSE TOGGLE
1057

FIG. 1A–2

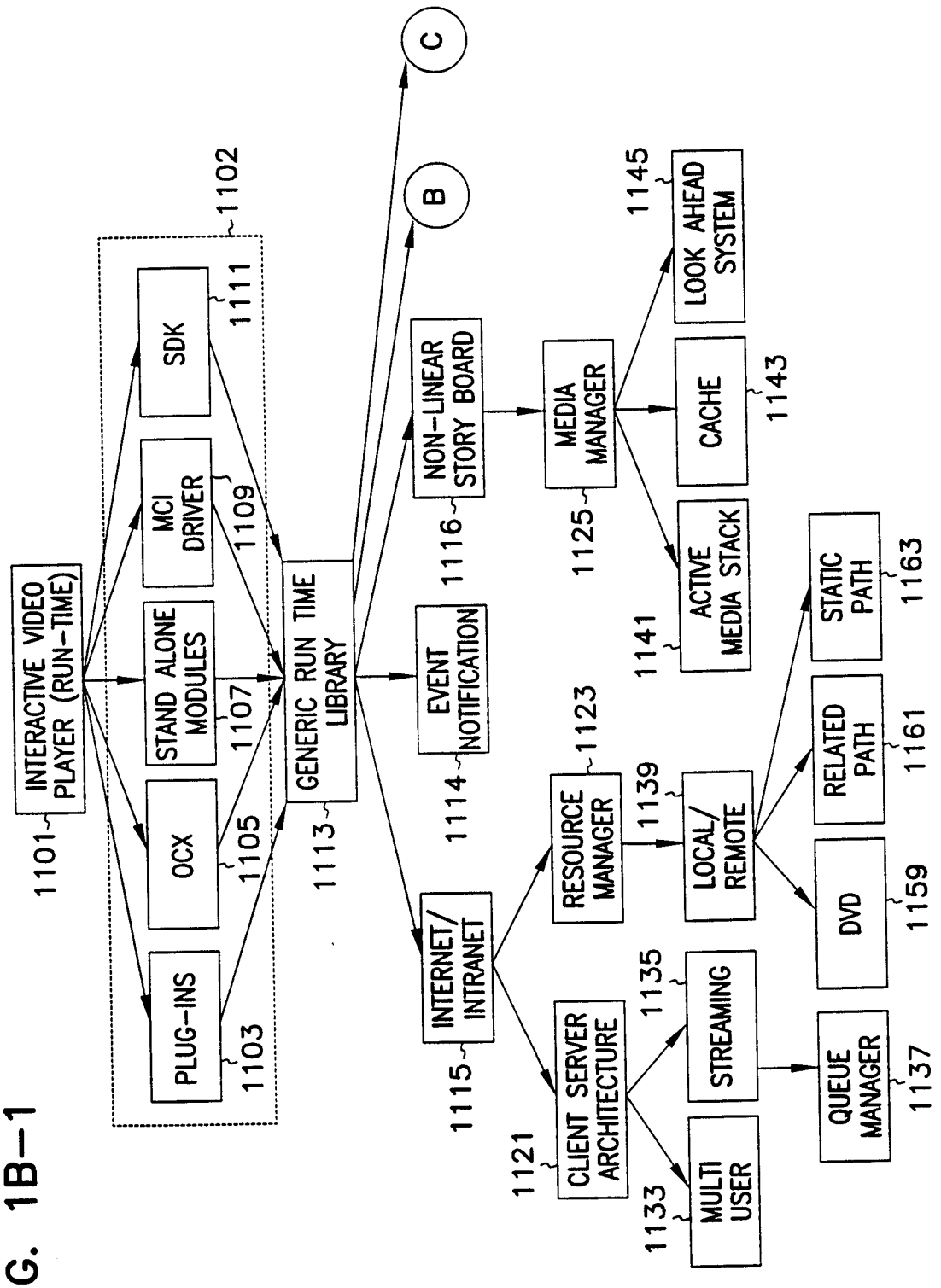
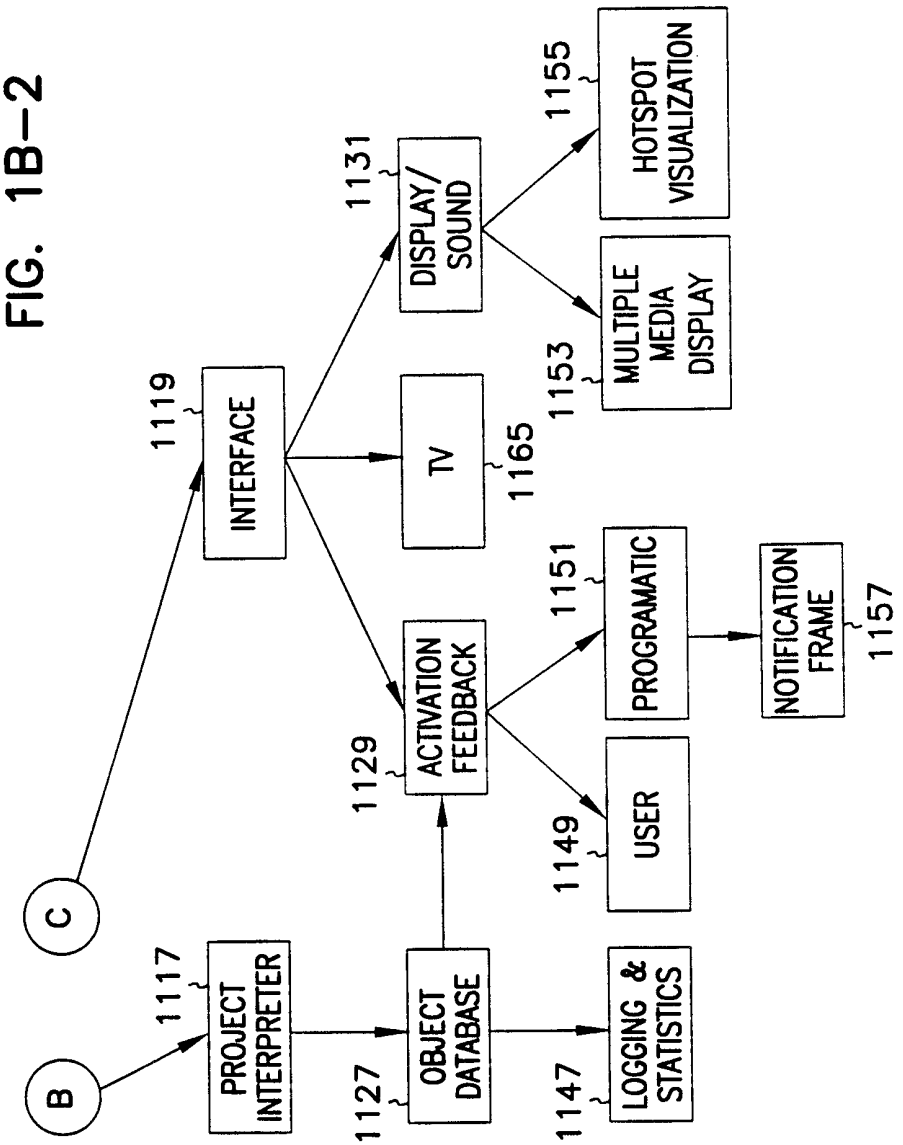

INTERACTIVE VIDEO PLAYER (RUN-TIME)
1101
1102
SDK
1111
MCI DRIVER
1109
STAND ALONE MODULES
1107
OCX
1105
PLUG-INS
1103
GENERIC RUN TIME LIBRARY
1113
C
B
NON-LINEAR STORY BOARD
1116
EVENT NOTIFICATION
1114
INTERNET/ INTRANET
1115
MEDIA MANAGER
1125
LOOK AHEAD SYSTEM
1145
CACHE
1143
ACTIVE MEDIA STACK
1141
RESOURCE MANAGER
1123
LOCAL/ REMOTE
1139
STATIC PATH
1163
RELATED PATH
1161
DVD
1159
CLIENT SERVER ARCHITECTURE
1121
STREAMING
1135
QUEUE MANAGER
1137
MULTI USER
1133

FIG. 1B-1

FIG. 1B–2


C
B
1119
INTERFACE
1117
PROJECT INTERPRETER
1127
OBJECT DATABASE
1129
ACTIVATION FEEDBACK
1131
DISPLAY/ SOUND
1165
TV
1149
USER
1147
LOGGING & STATISTICS
1155
HOTSPOT VISUALIZATION
1153
MULTIPLE MEDIA DISPLAY
1151
PROGRAMATIC
1157
NOTIFICATION FRAME

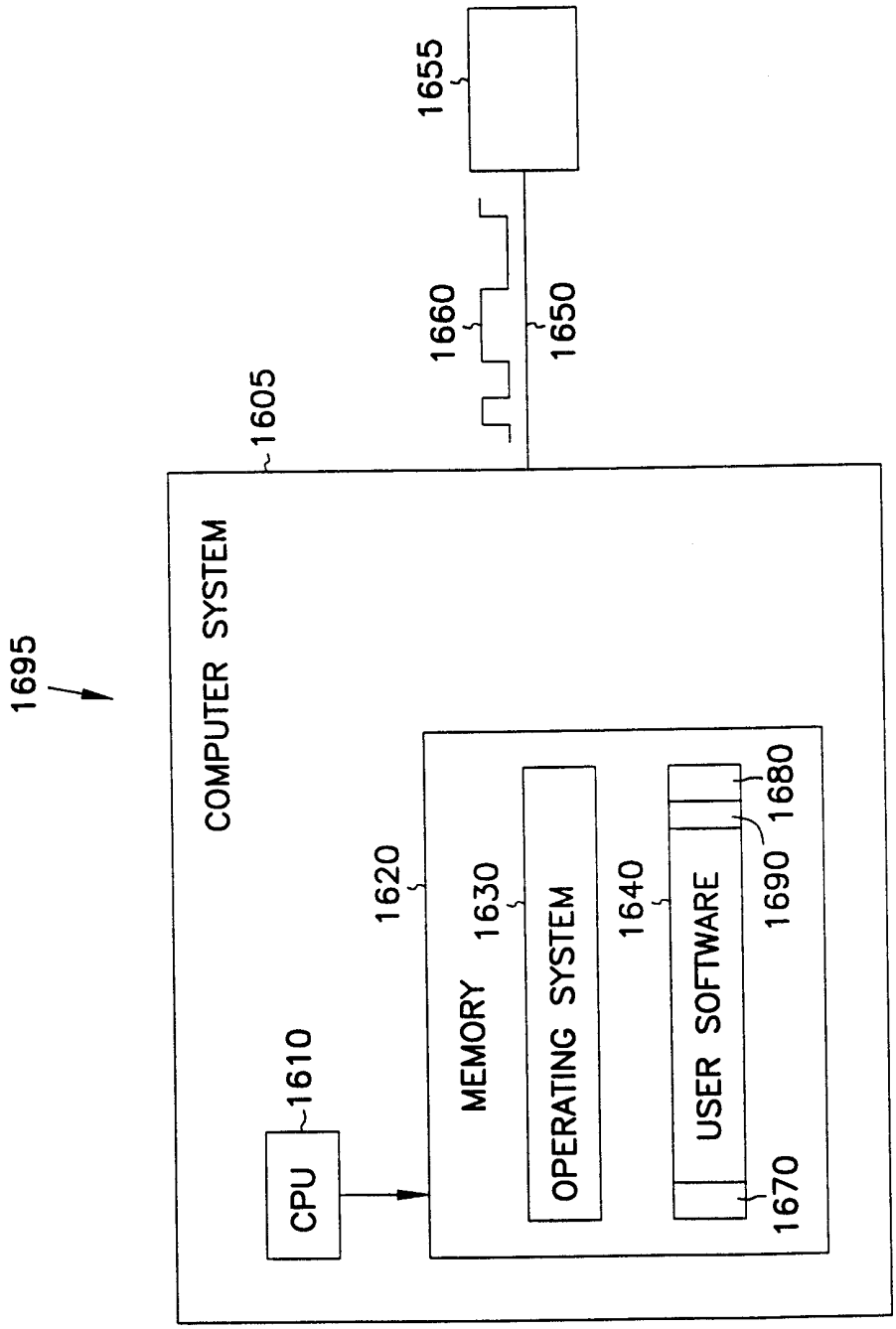

1695
COMPUTER SYSTEM
1605
1655
1660
1650
1620
MEMORY
1630
OPERATING SYSTEM
1640
USER SOFTWARE
1690
1680
1670
CPU
1610

FIG. 1C

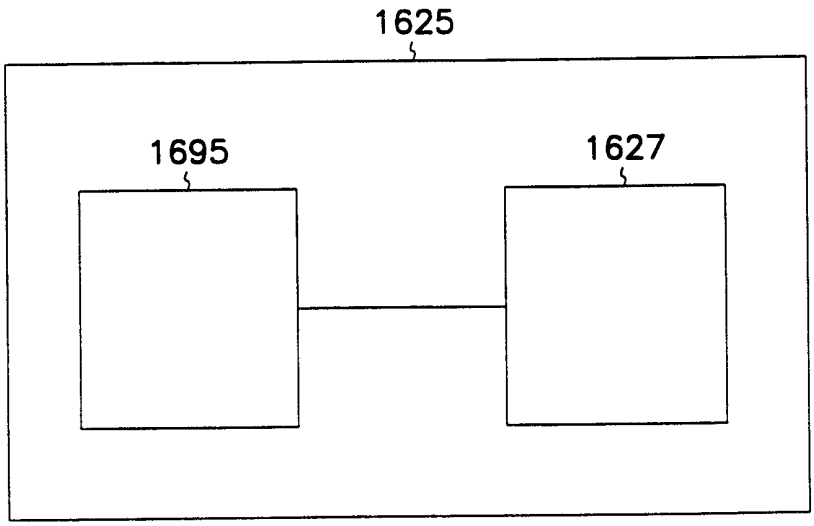


FIG. 1D

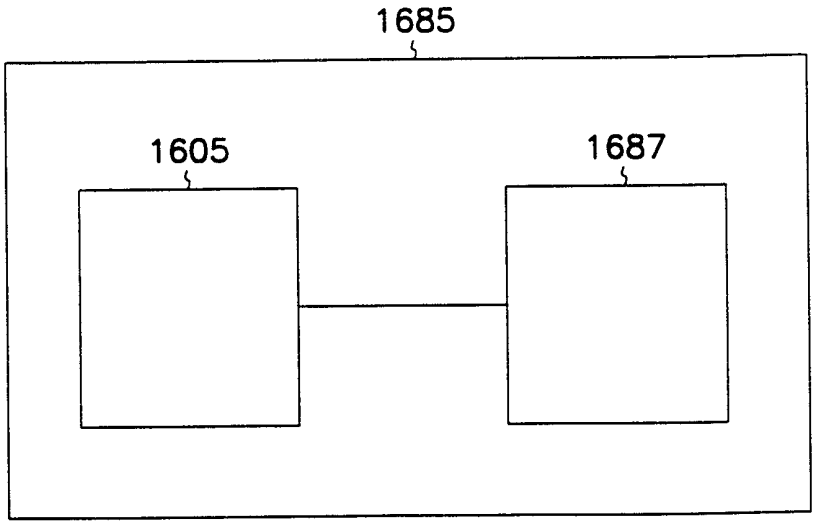


FIG. 1E

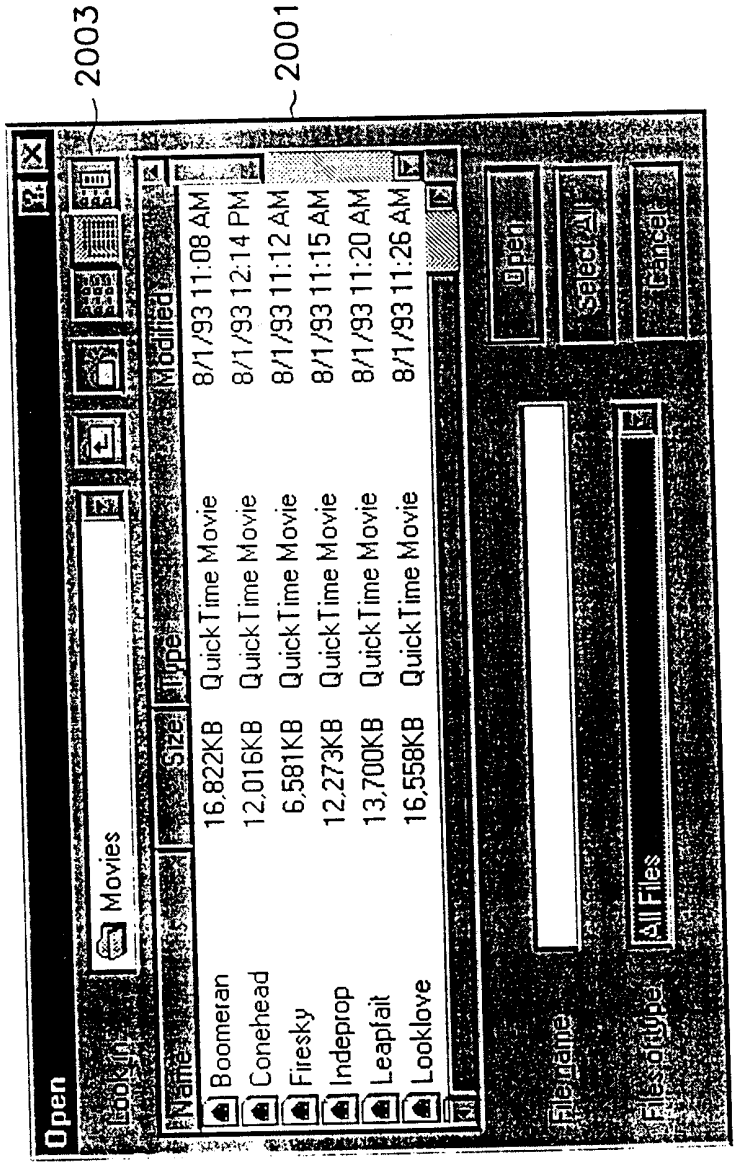

2003
2001
Open
Look in:
Movies
Name
Size
Type
Modified
Boomeran 16,822KB QuickTime Movie 8/1/93 11:08 AM
Conehead 12,016KB QuickTime Movie 8/1/93 12:14 PM
Firesky 6,581KB QuickTime Movie 8/1/93 11:12 AM
Indeprop 12,273KB QuickTime Movie 8/1/93 11:15 AM
Leapfait 13,700KB QuickTime Movie 8/1/93 11:20 AM
Looklove 16,558KB QuickTime Movie 8/1/93 11:26 AM
Filename:
Files of type:
All Files
Open
Select All
Cancel

FIG. 2

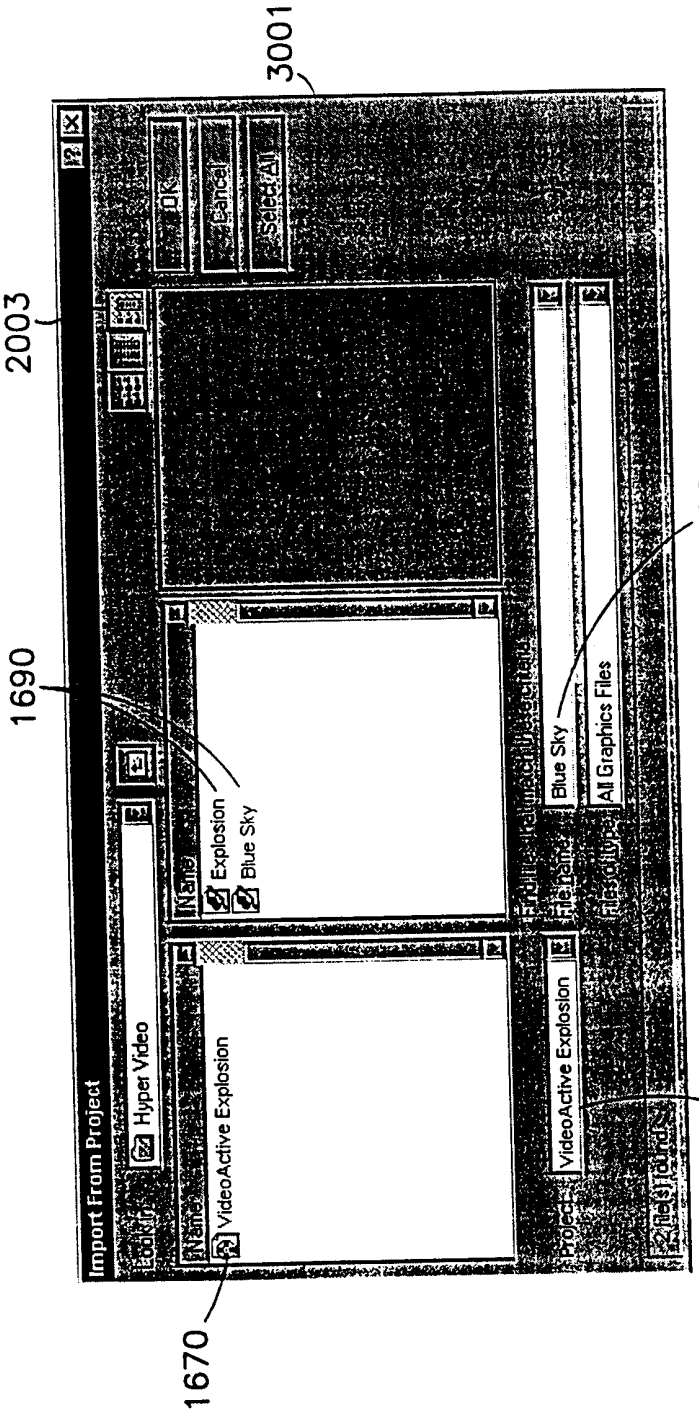


FIG. 3

COPYRIGHT EPHYX TECHNOLOGIES, LTD. (ISRAEL)

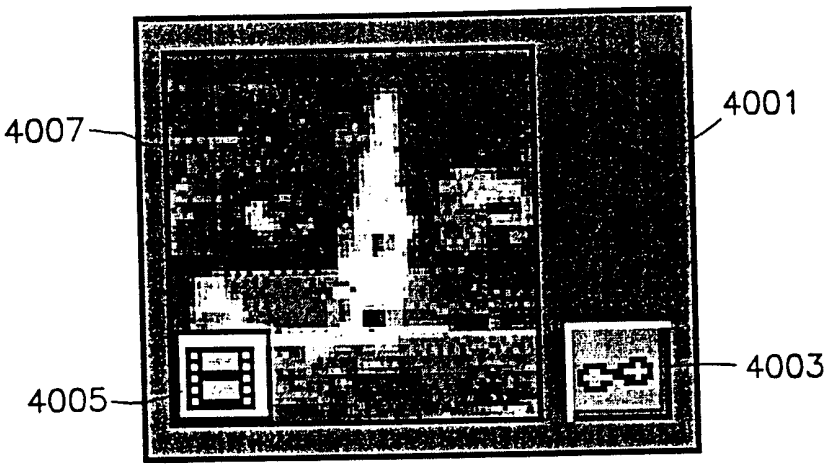


FIG. 4

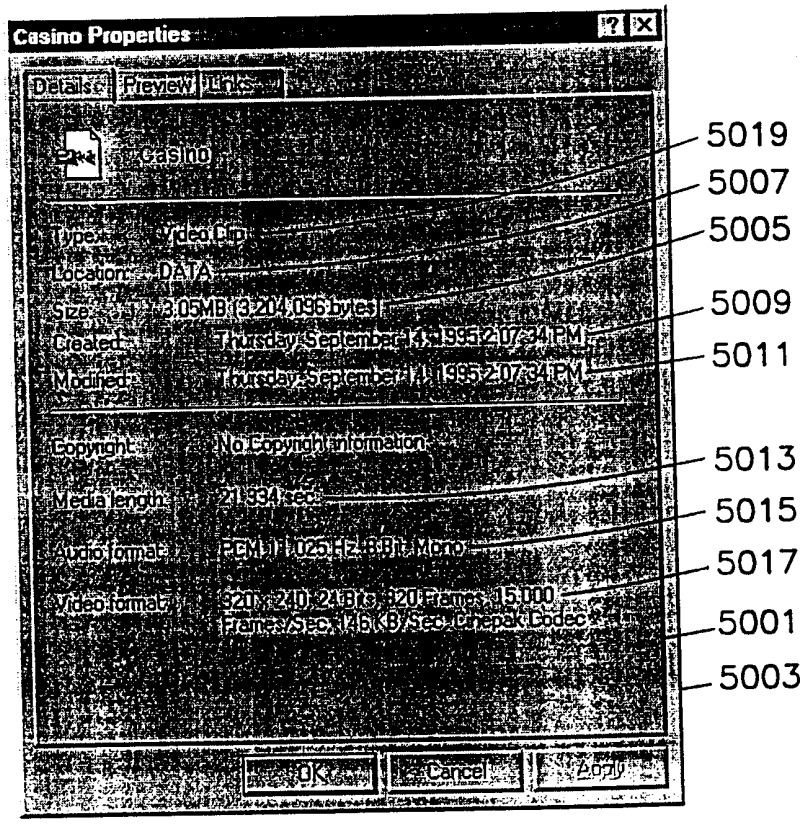

Casino Properties
Details
Preview
Links
Casino
Type:
Video Clip
Location:
DATA
Size:
3.05MB (3,204,096 bytes)
Created:
Thursday, September 14, 1995 2:07:34 PM
Modified:
Thursday, September 14, 1995 2:07:34 PM
Copyright:
No Copyright Information
Media length:
21.934 sec.
Audio format:
PCM, 11.025 kHz, 8 bit, Mono
Video format:
320 x 240, 24 Bits, 20 Frames, 15.000 Frames/Sec, 146 KB/Sec, Cinepak Codec
OK
Cancel
Apply
5019
5007
5005
5009
5011
5013
5015
5017
5001
5003

FIG. 5

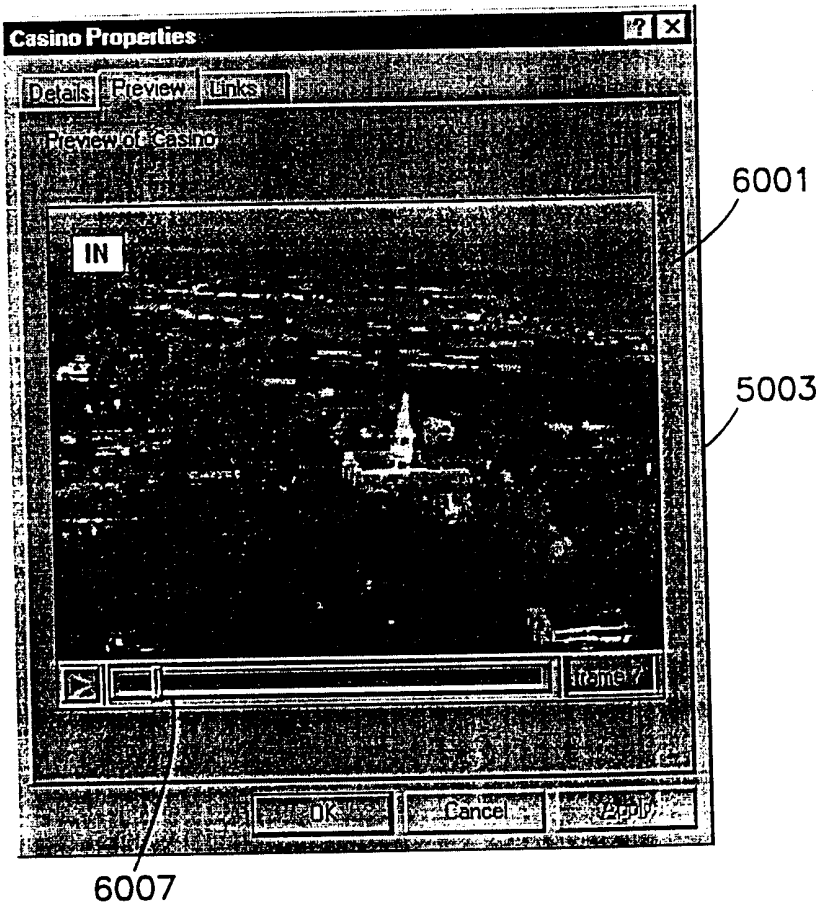

Casino Properties
Details
Preview
Links
Preview of casino
IN
Frame 7
OK
Cancel
Apply
6001
5003
6007

FIG. 6

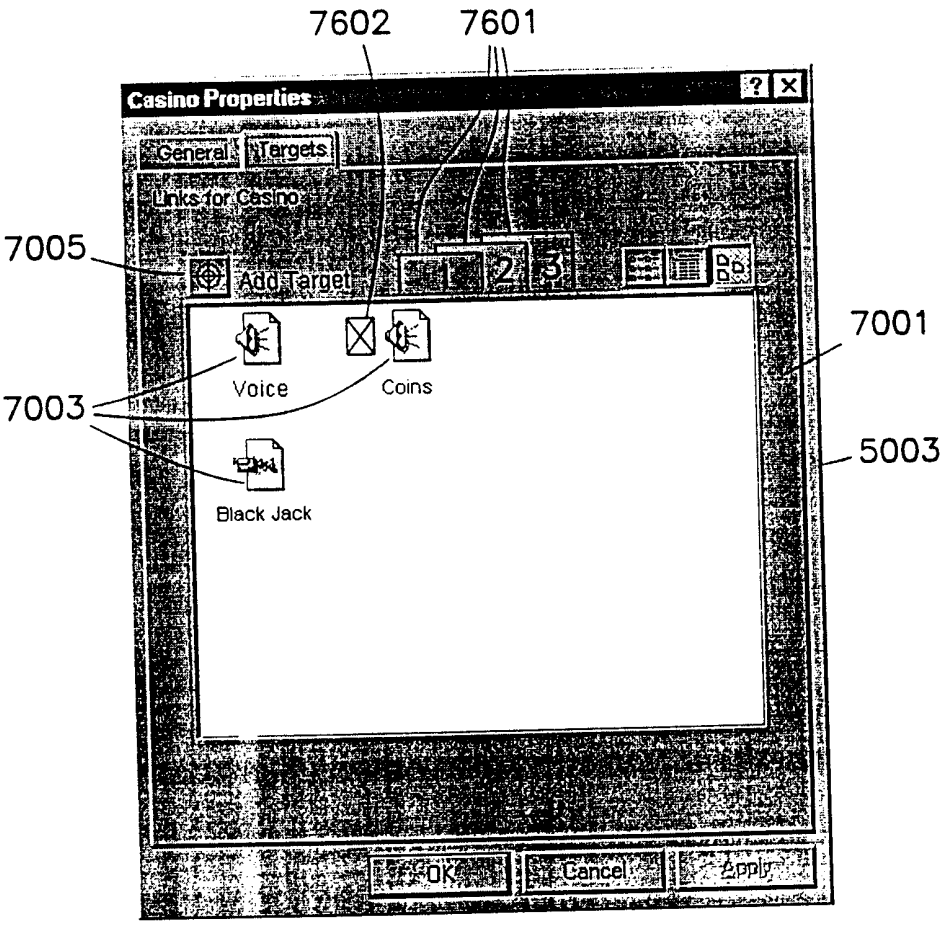

7602
7601
Casino Properties
General
Targets
Links for Casino
7005
Add Target
1
2
3
7001
Voice
Coins
7003
5003
Black Jack
OK
Cancel
Apply

FIG. 7A

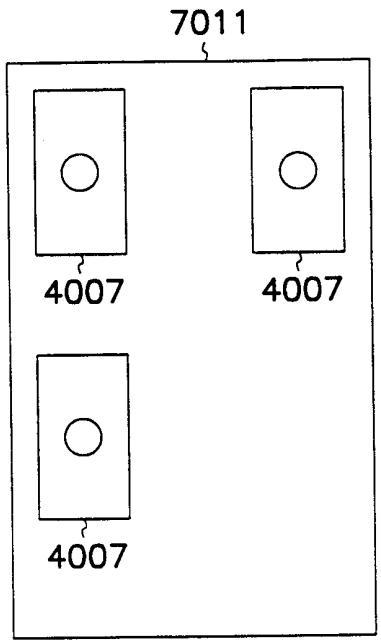


FIG. 7B

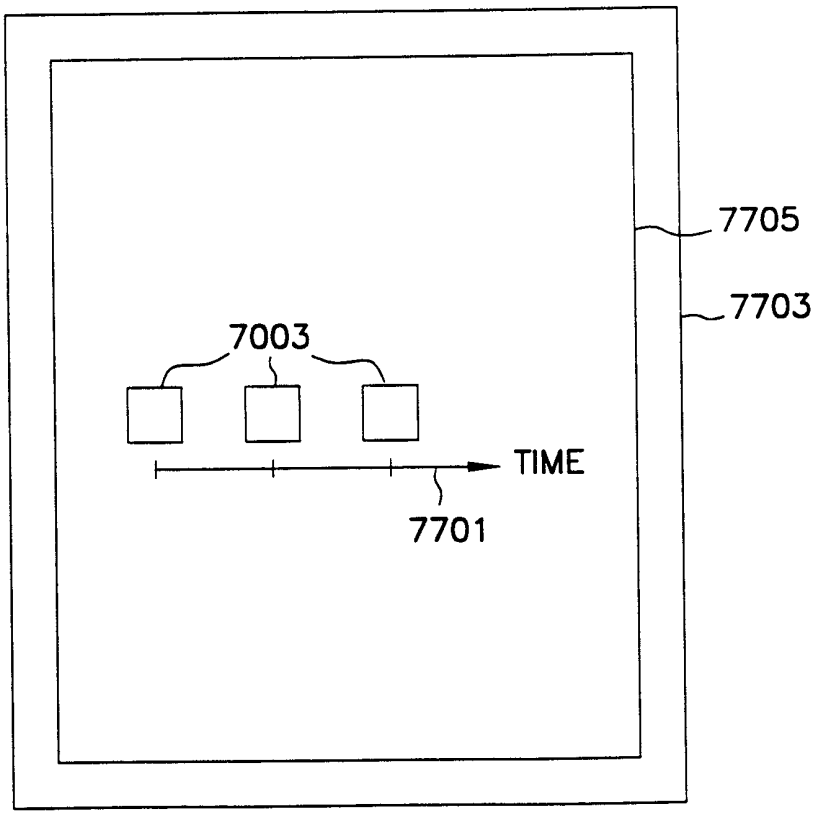


FIG. 7D

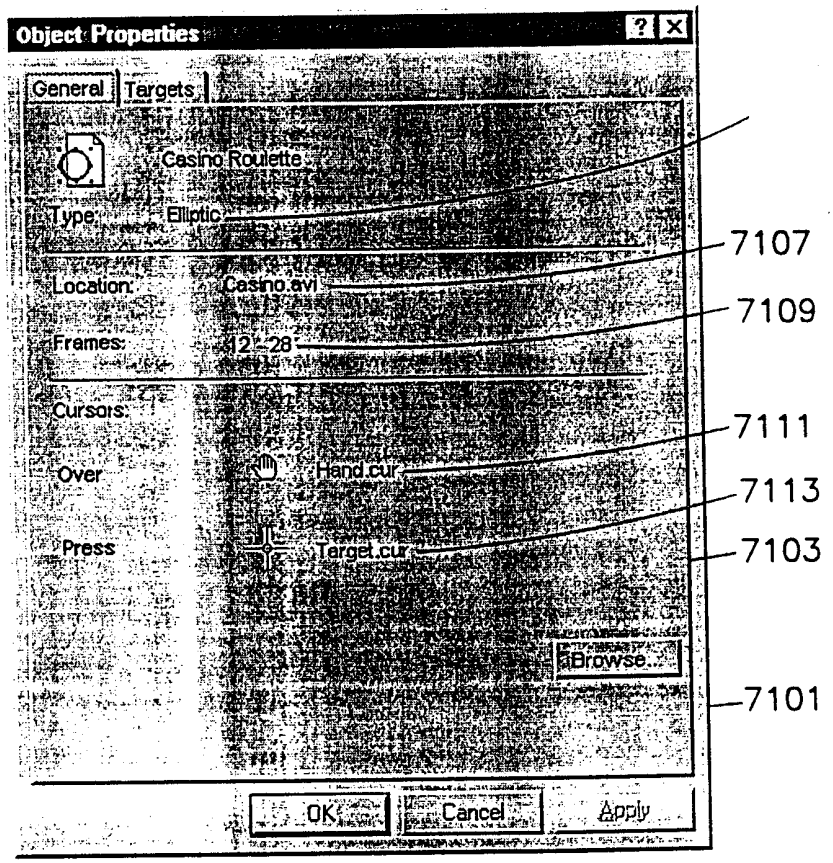

Object Properties
General
Targets
Casino Roulette
Type:
Elliptic
Location:
Casino.avi
Frames:
1
2
28
Cursors:
Over
Hand.cur
Press
Target.cur
Browse...
OK
Cancel
Apply
7107
7109
7111
7113
7103
7101

FIG. 7C

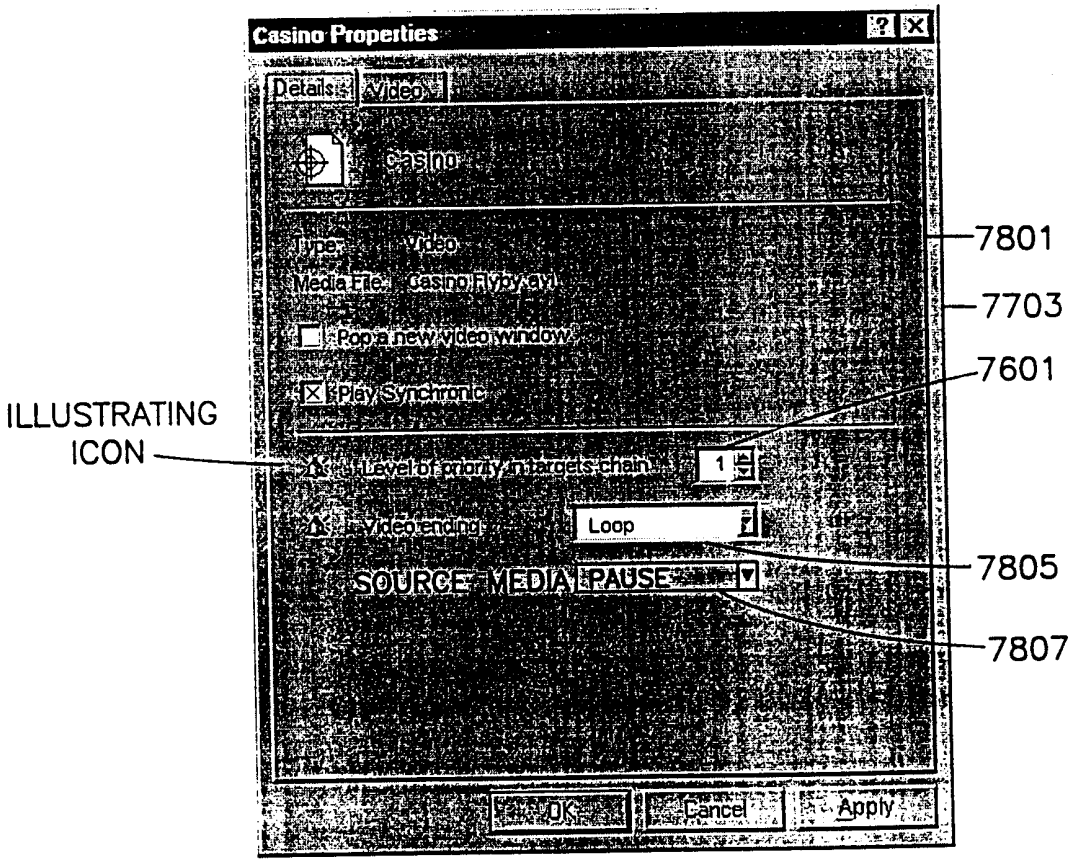

Casino Properties
Details
Video
Casino
Type: Video
Media File: casino Flyby.avi
Pop a new video window
Play Synchronic
Level of priority in targets chain
1
Video ending
Loop
SOURCE MEDIA
PAUSE
OK
Cancel
Apply
ILLUSTRATING ICON
7801
7703
7601
7805
7807

FIG. 7E

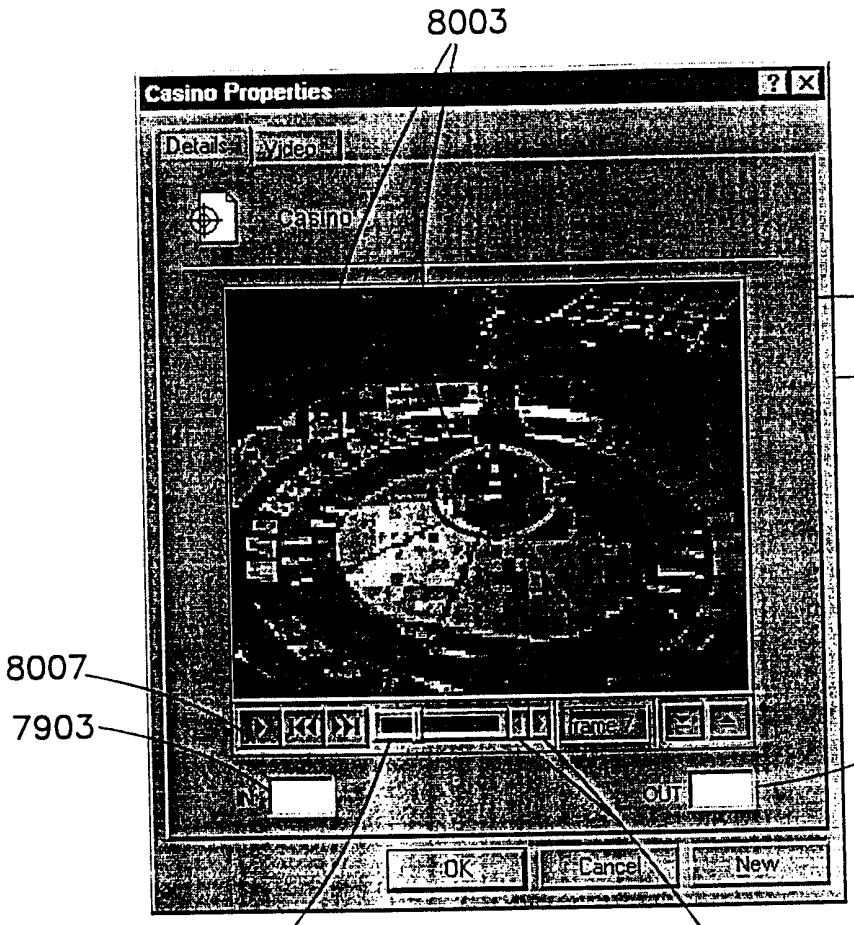

8003
Casino Properties
Details
Video
Casino
7901
7703
8007
7903
7905
frame 7
IN
OUT
OK
Cancel
New
6007
8001

FIG. 7F

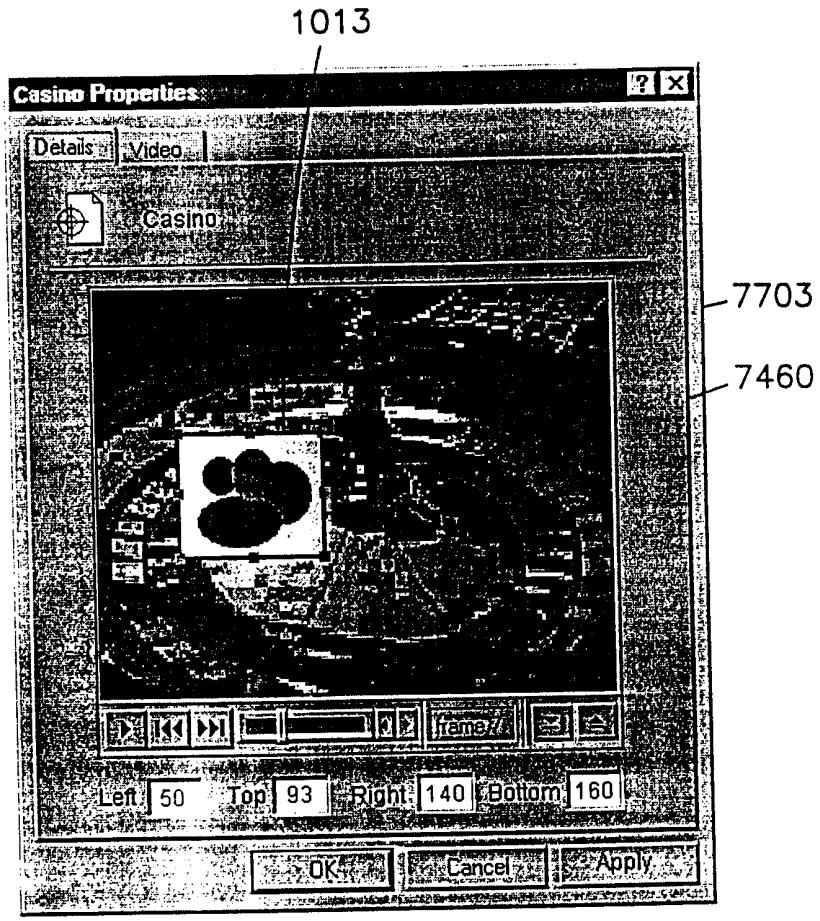

1013
Casino Properties
Details
Video
Casino
7703
7460
frame
Left 50
Top 93
Right 140
Bottom 160
OK
Cancel
Apply

FIG. 7G

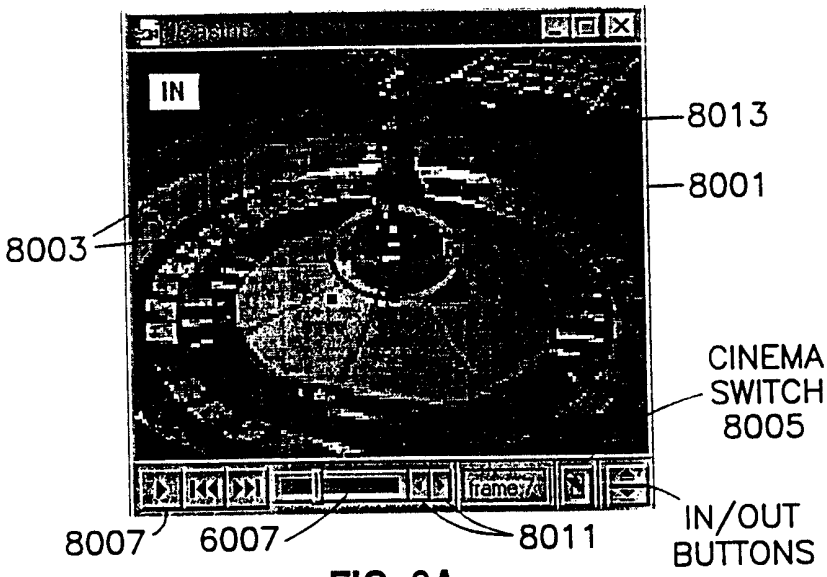

IN
8013
8001
8003
CINEMA
SWITCH
8005
IN/OUT
BUTTONS
8007
6007
8011

FIG. 8A

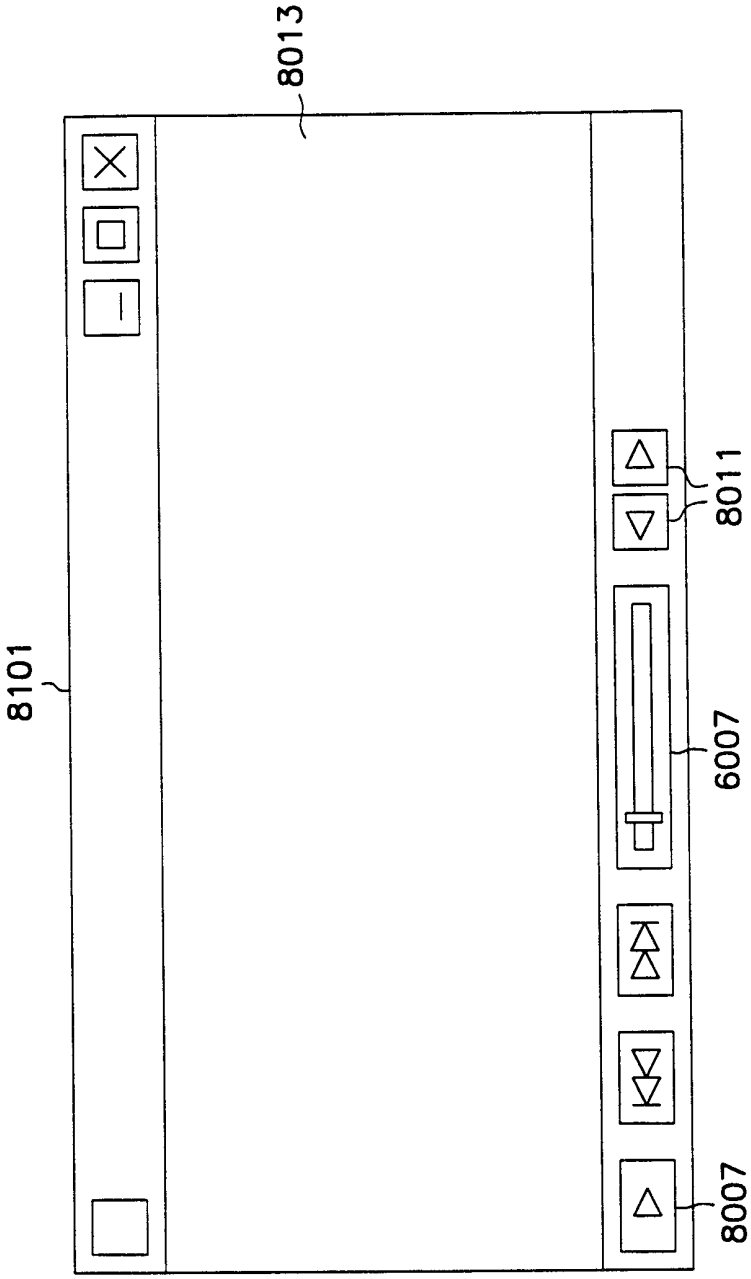


FIG. 8B

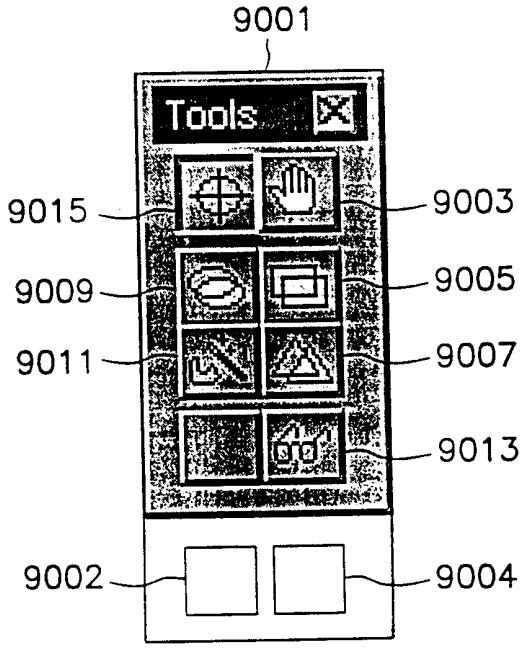


FIG. 9A

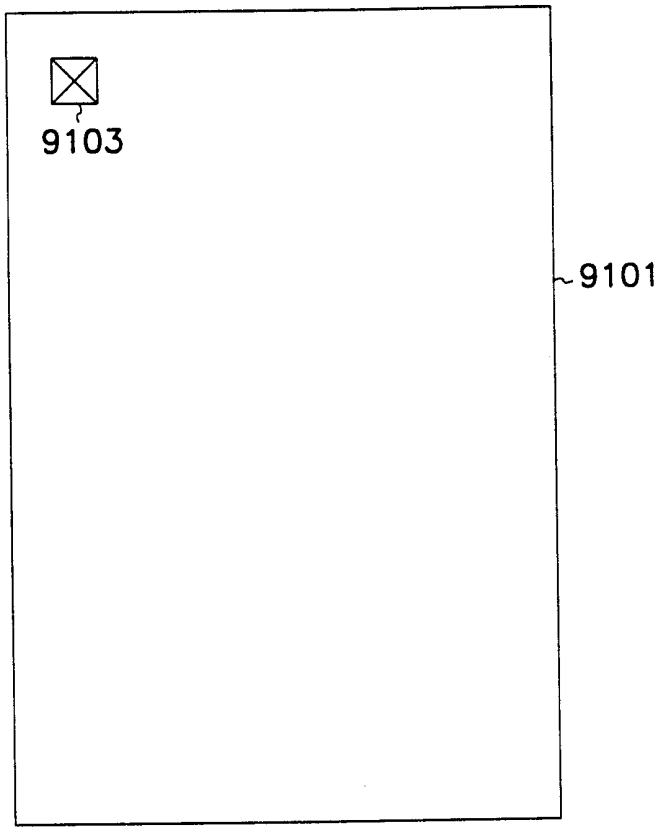


FIG. 9B

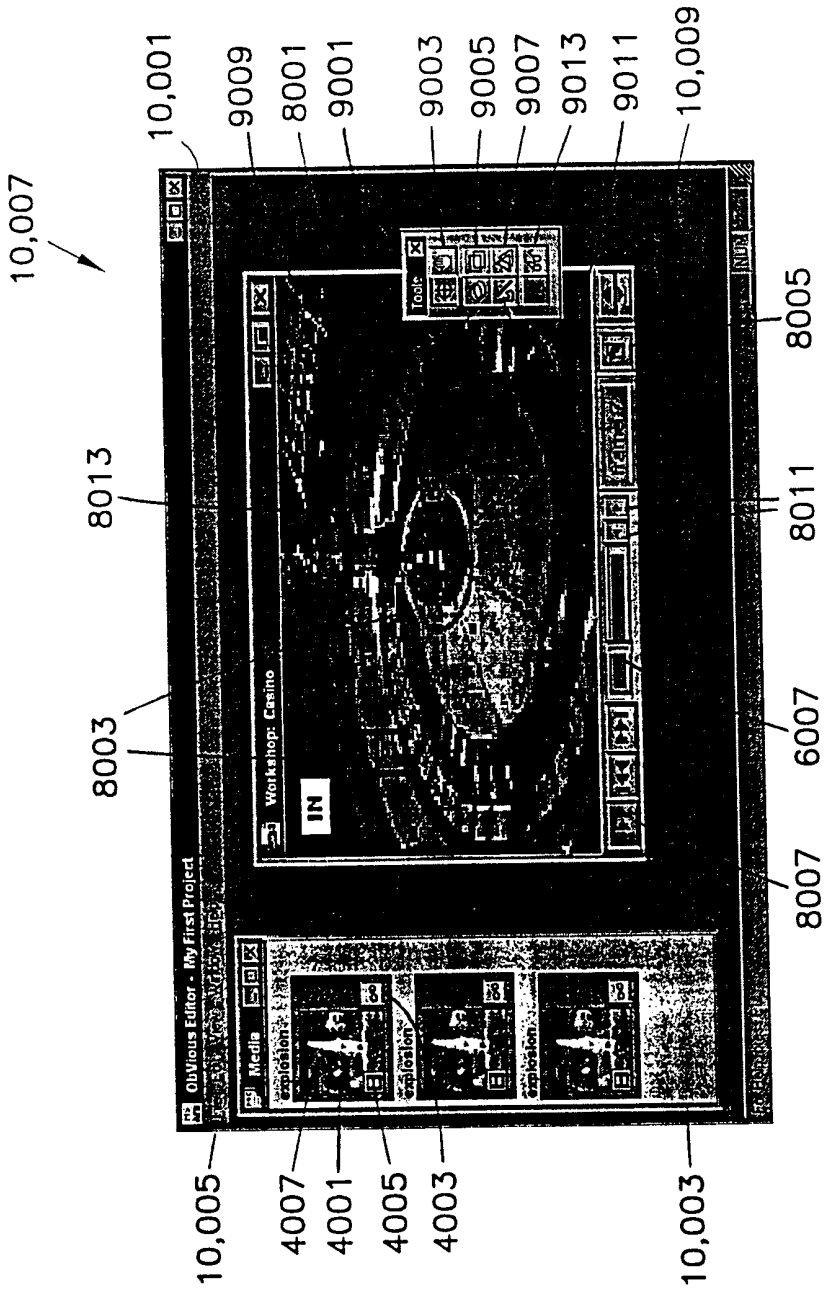

10,007
10,001
9009
8001
9001
9003
9005
9007
9013
9011
10,009
8005
8013
8011
8003
6007
8007
10,005
4007
4001
4005
4003
10,003
ObVious Editor - My First Project
Media
Workshop: Casino
IN
Tools

FIG. 10

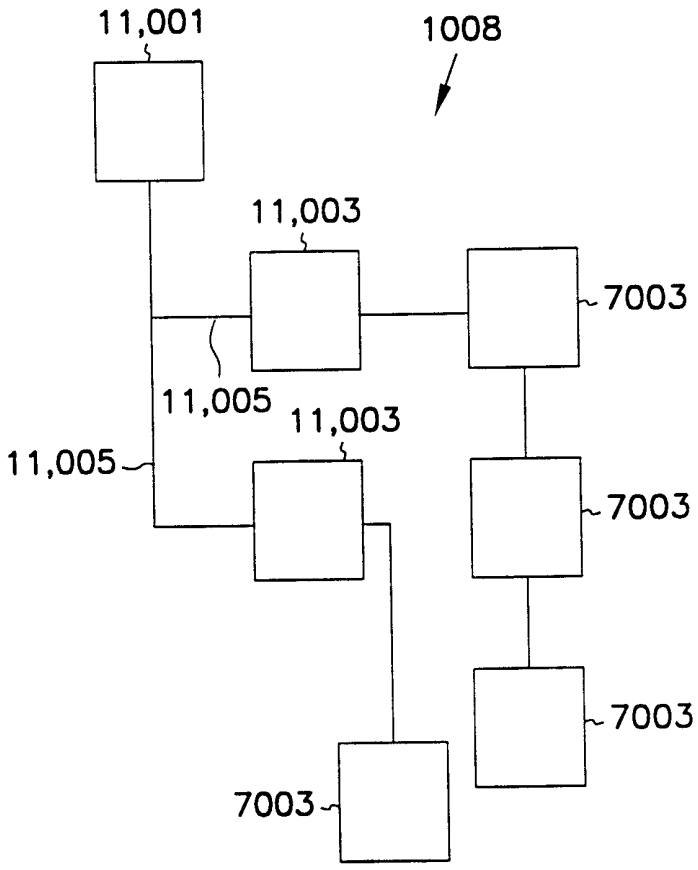


FIG. 11

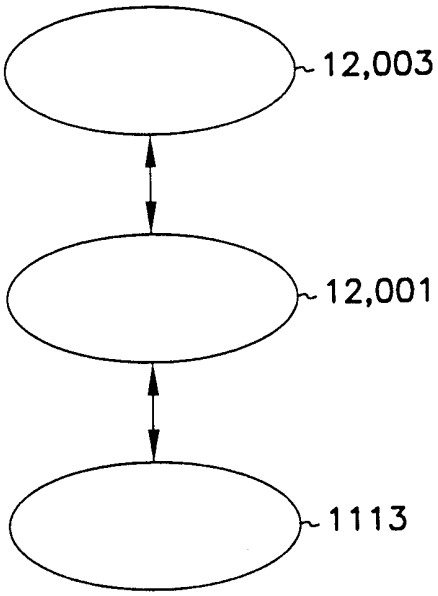


FIG. 12

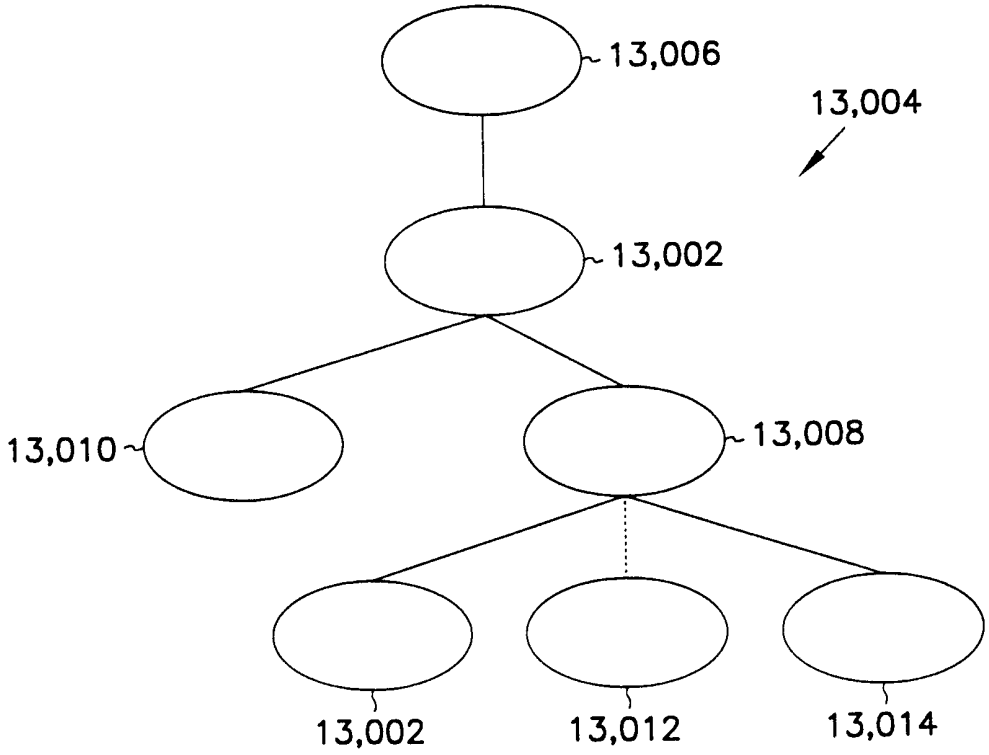


FIG. 13

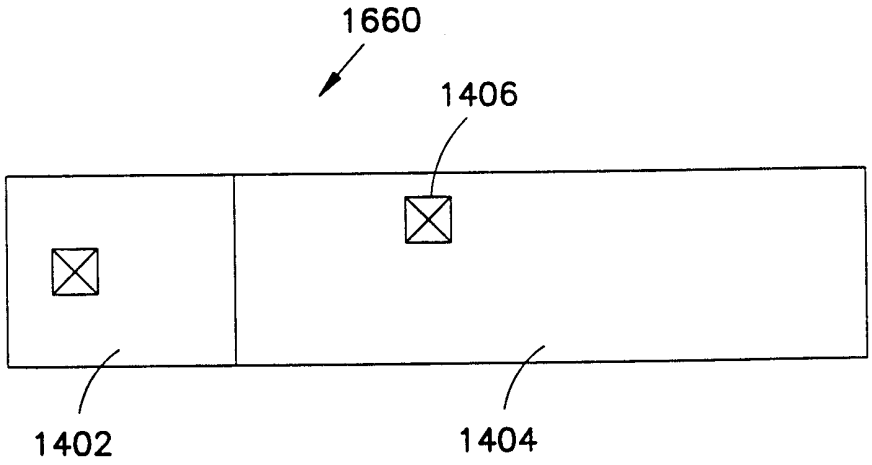


FIG. 14

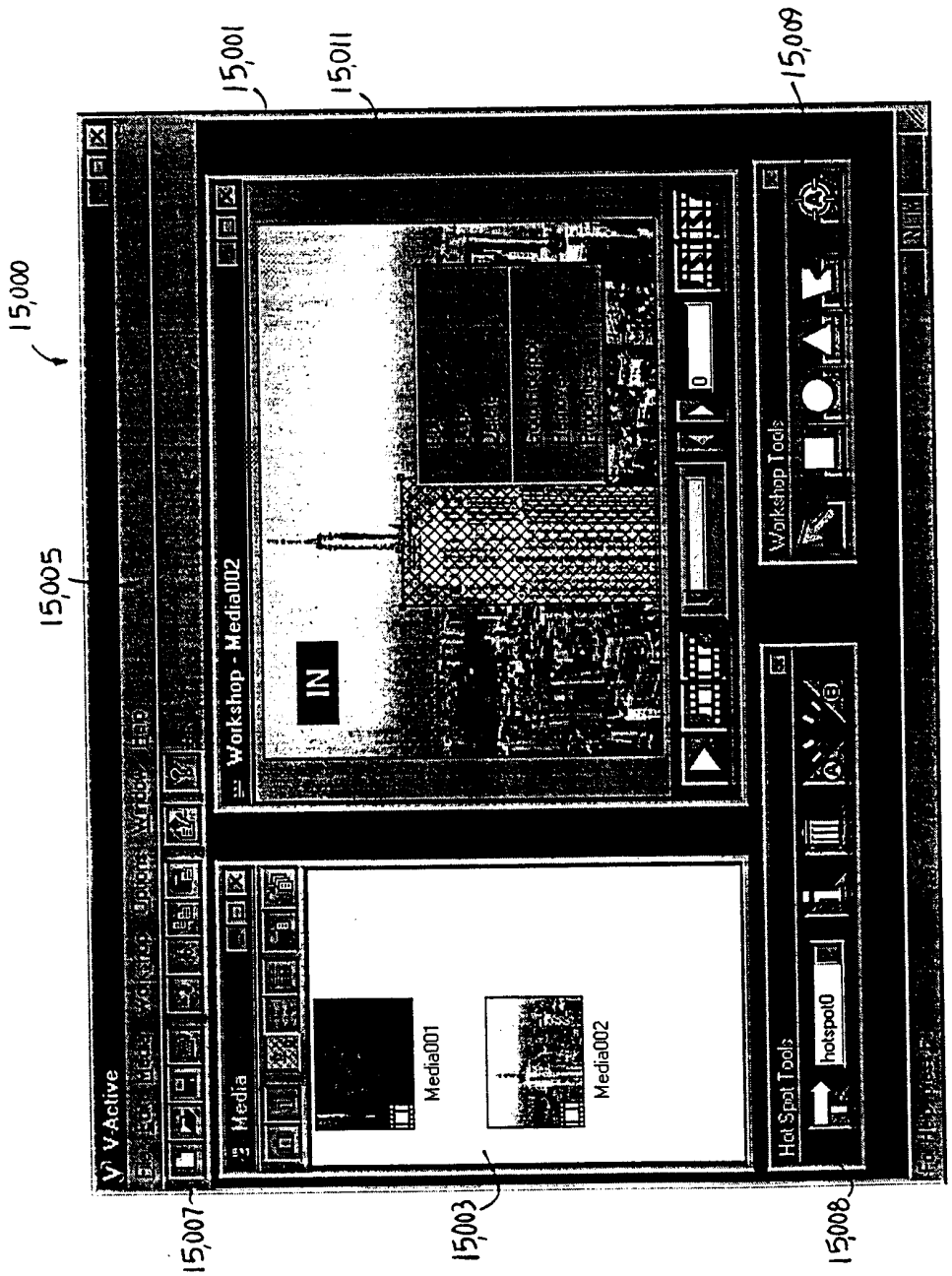

15,000
15,001
15,011
15,009
15,005
15,007
15,003
15,008
V Active
Media
Media001
Media002
Workshop - Media002
IN
Workshop Tools
Hot Spot Tools
hotspot0

FIG. 15

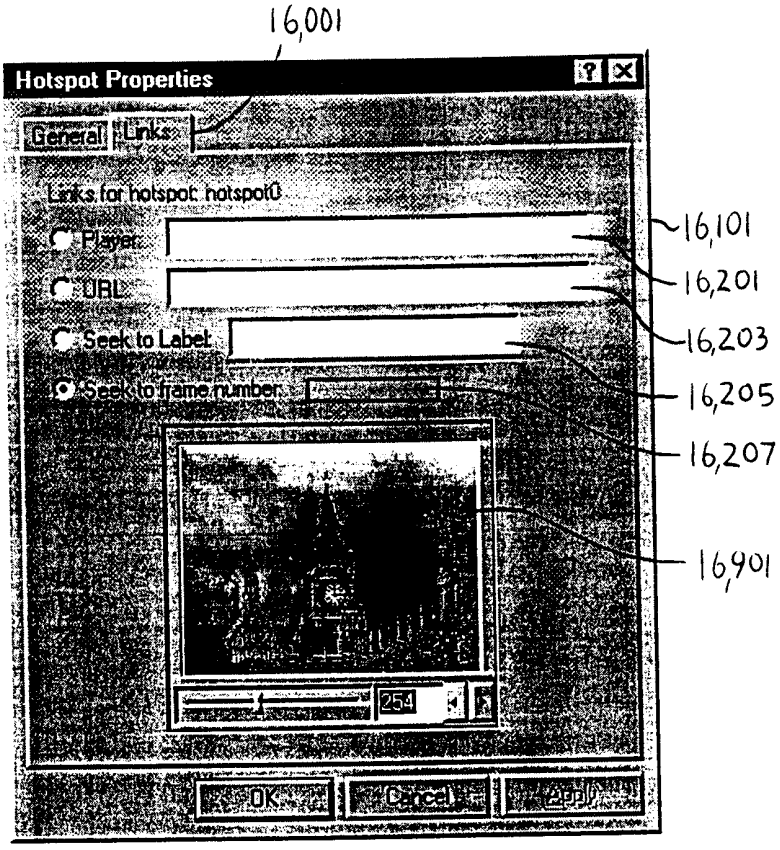

16,001
Hotspot Properties
General
Links
Links for hotspot hotspot0
Player
URL
Seek to Label
Seek to frame number
254
OK
Cancel
Apply
16,101
16,201
16,203
16,205
16,207
16,901

FIG. 16

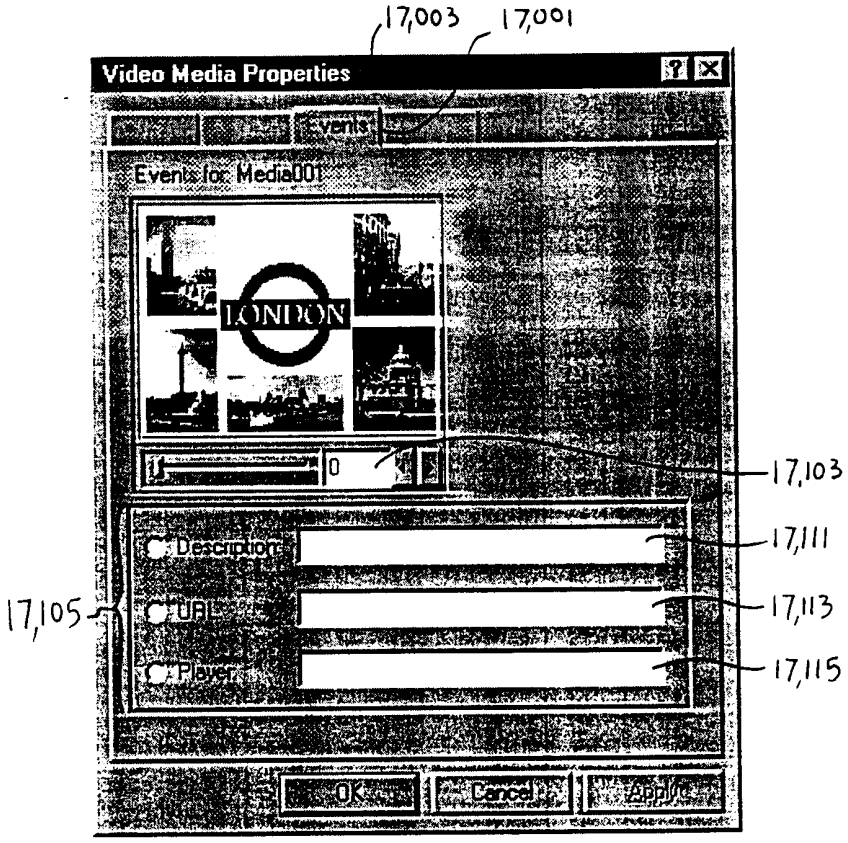

17,003
17,001
Video Media Properties
Events
Events for Media001
LONDON
0
17,103
Description:
17,111
17,105
URL:
17,113
Player:
17,115
OK
Cancel
Apply

FIG. 17

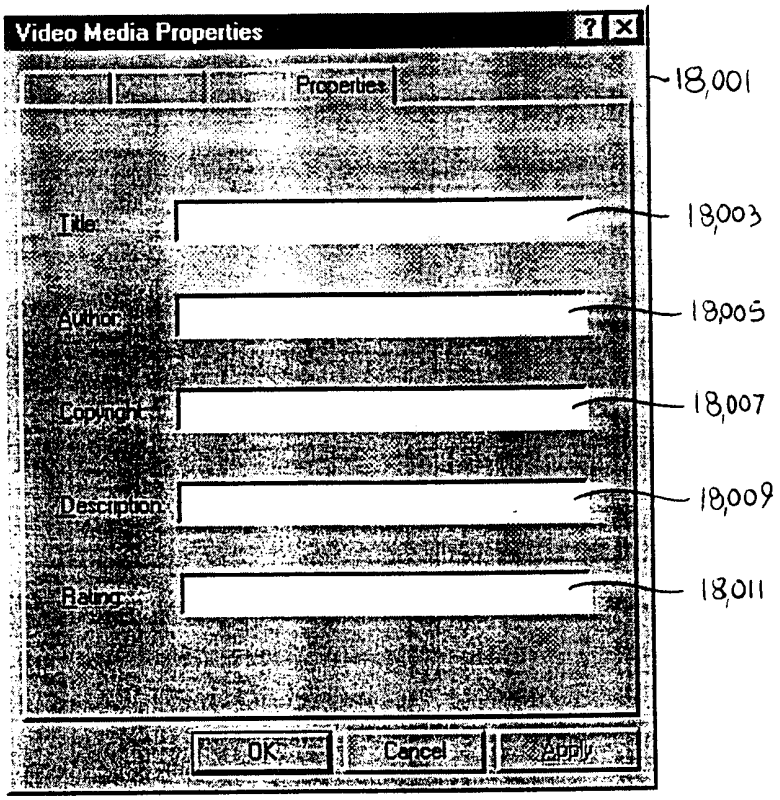

Video Media Properties
Properties
18,001
Title:
18,003
Author:
18,005
Copyright:
18,007
Description:
18,009
Rating:
18,011
OK
Cancel
Apply

FIG. 18

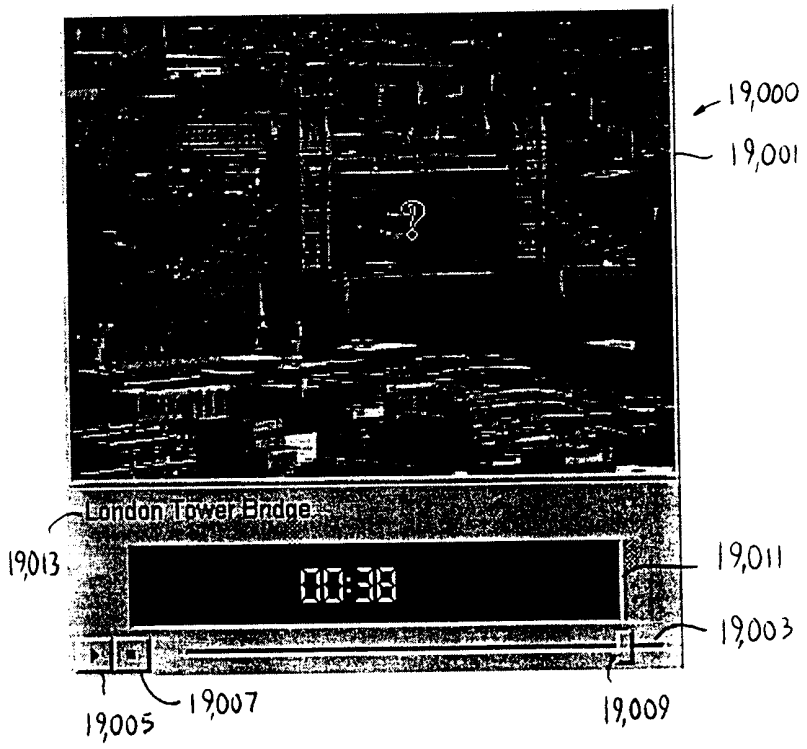

19,000
19,001
London Tower Bridge
19,013
08:38
19,011
19,003
19,005
19,007
19,009

FIG. 19

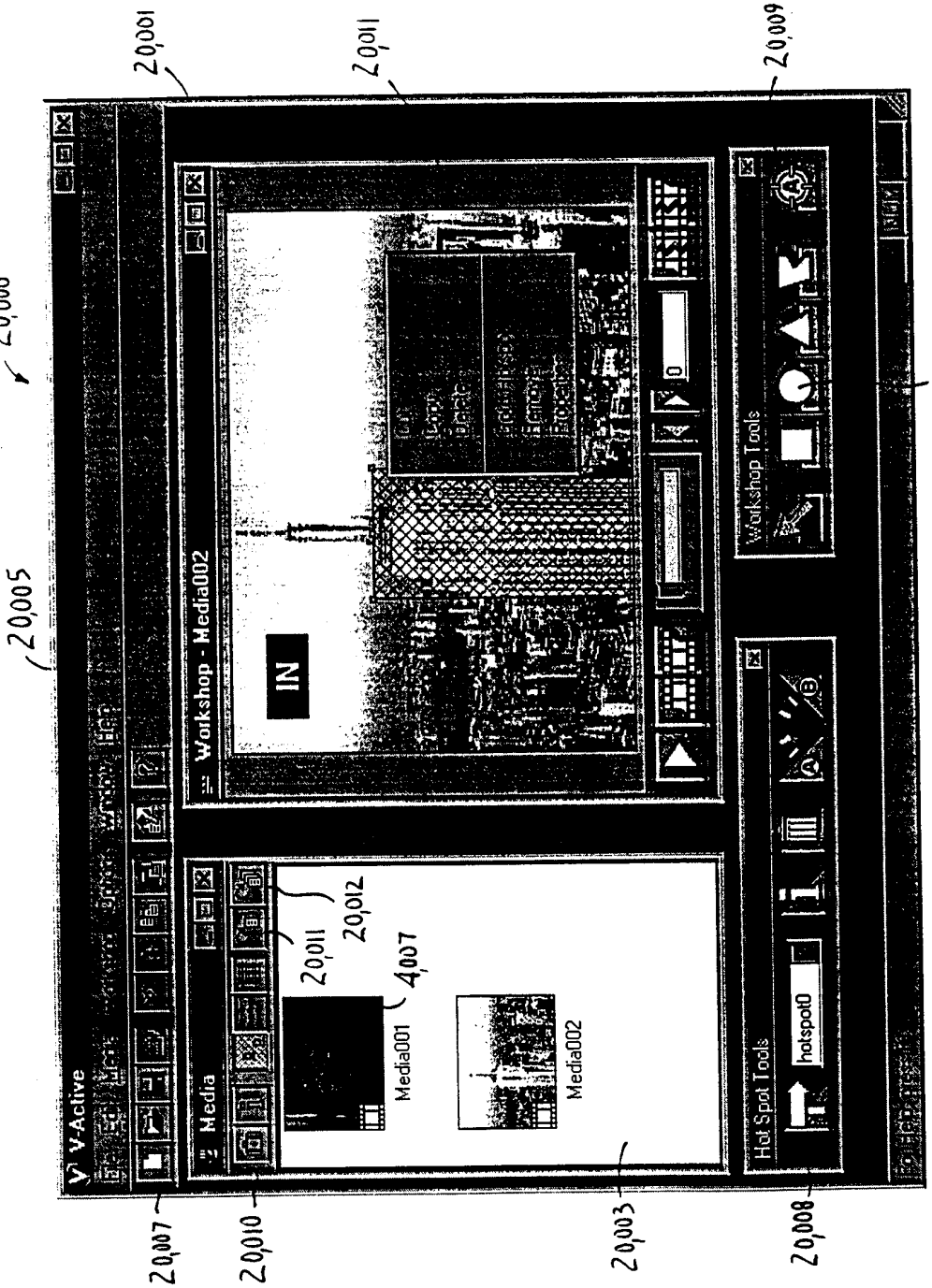

20,000
20,001
20,011
20,009
20,005
20,101
20,007
20,010
20,003
20,008
V-Active
Workshop - Media002
IN
Media
20,011
20,012
4,007
Media001
Media002
Hot Spot Tools
hotspot0
Workshop Tools

FIG. 20

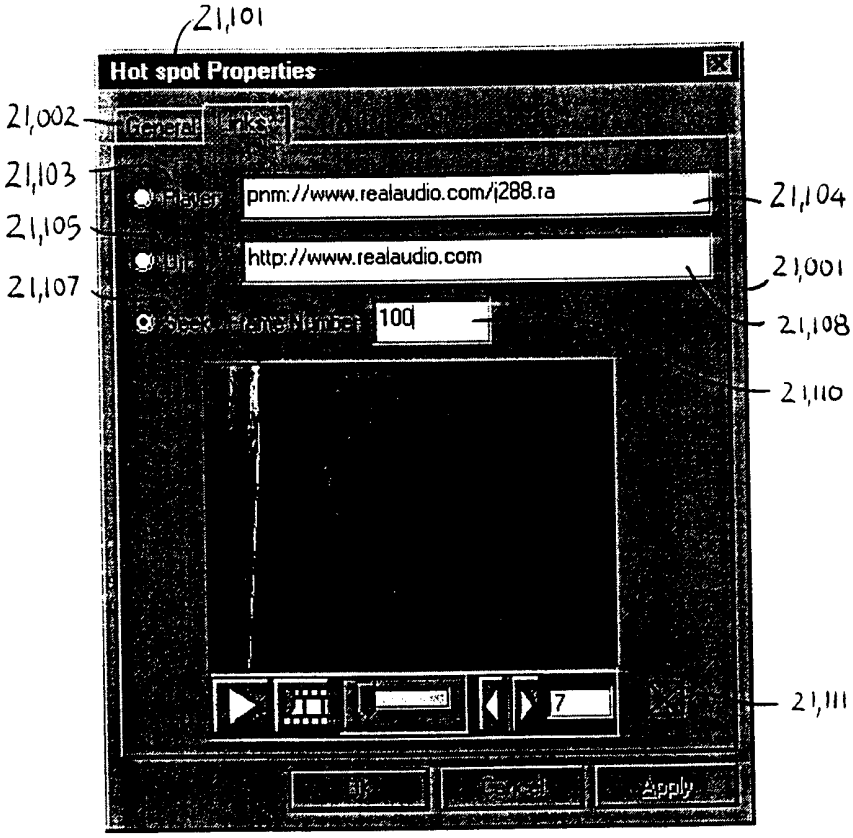

21,101
Hot spot Properties
21,002
General
21,103
21,105
21,107
pnm://www.realaudio.com/j288.ra
21,104
http://www.realaudio.com
21,001
100
21,108
21,110
7
21,111
OK
Cancel
Apply

FIG. 21

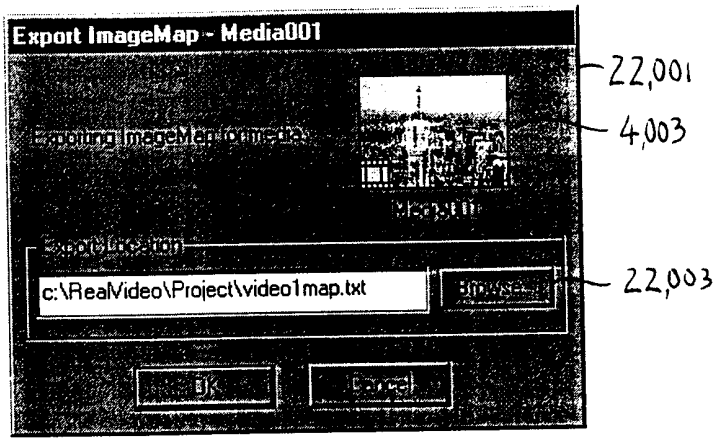

Export ImageMap - Media001
22,001
4,003
c:\RealVideo\Project\video1map.txt
22,003
OK
Cancel

FIG. 22

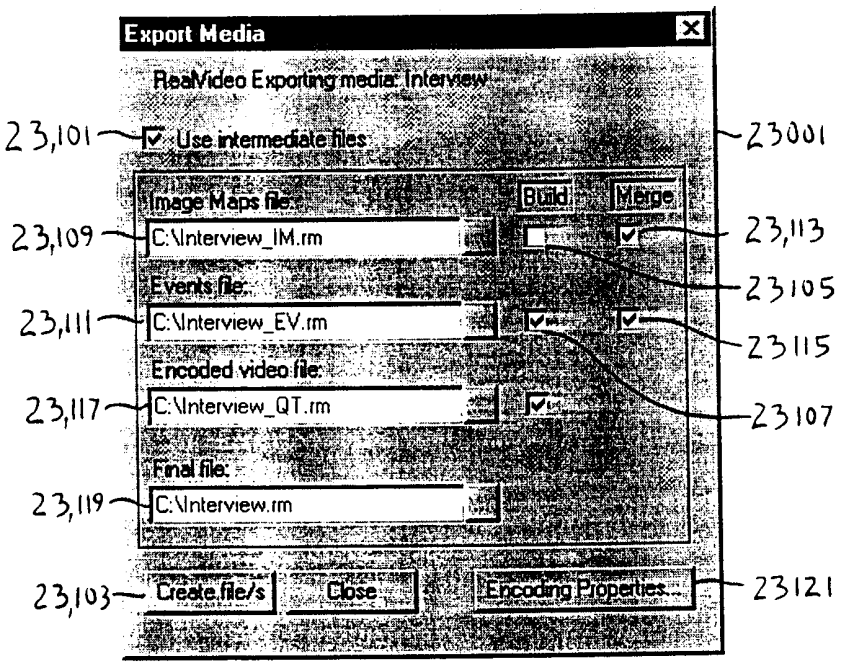

Export Media
RealVideo Exporting media: Interview
23,101
Use intermediate files
23001
Image Maps file:
Build
Merge
23,109
C:\Interview_IM.rm
23,113
23105
Events file:
23,111
C:\Interview_EV.rm
23115
Encoded video file:
23,117
C:\Interview_QT.rm
23107
Final file:
23,119
C:\Interview.rm
23,103
Create file/s
Close
Encoding Properties...
23121

FIG. 23

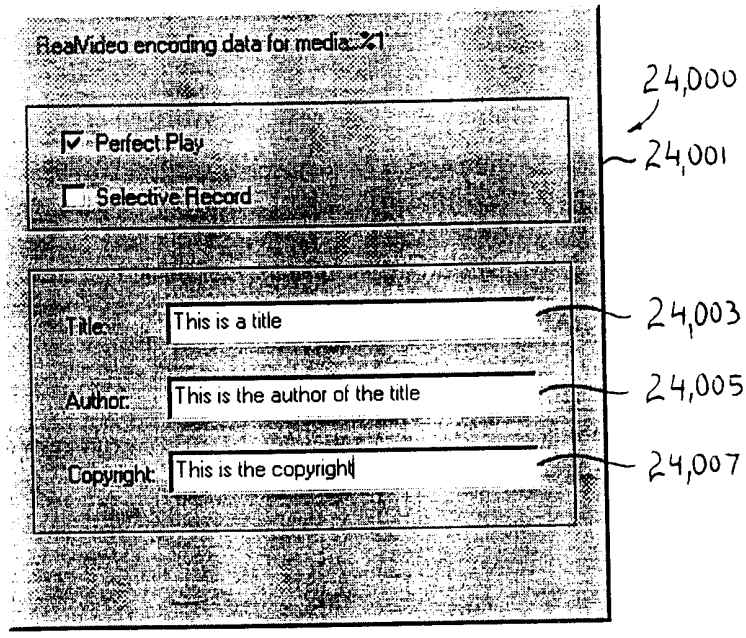

RealVideo encoding data for media: %1
Perfect Play
Selective Record
Title:
This is a title
Author:
This is the author of the title
Copyright:
This is the copyright
24,000
24,001
24,003
24,005
24,007

FIG. 24A

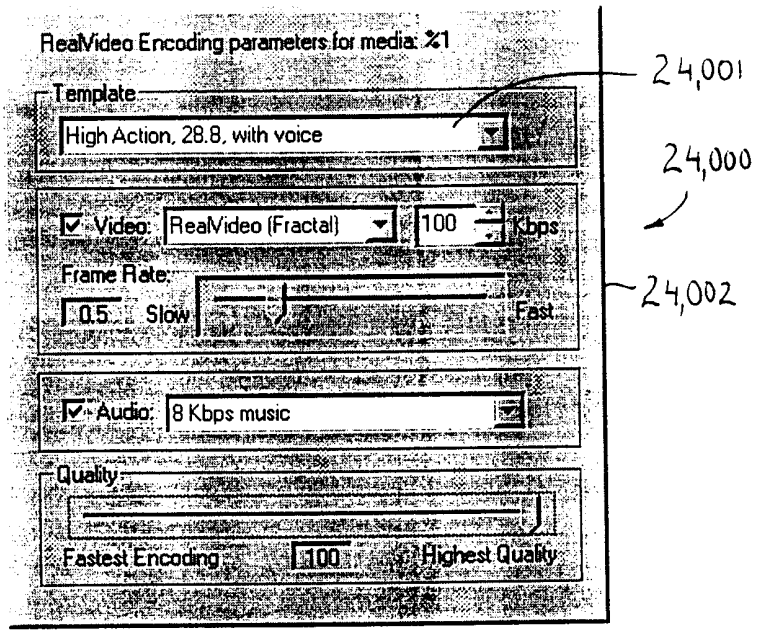

RealVideo Encoding parameters for media: #1
Template
High Action, 28.8, with voice
Video:
RealVideo (Fractal)
100
Kbps
Frame Rate:
0.5
Slow
Fast
Audio:
8 Kbps music
Quality
Fastest Encoding
100
Highest Quality
24,001
24,000
24,002

FIG. 24B

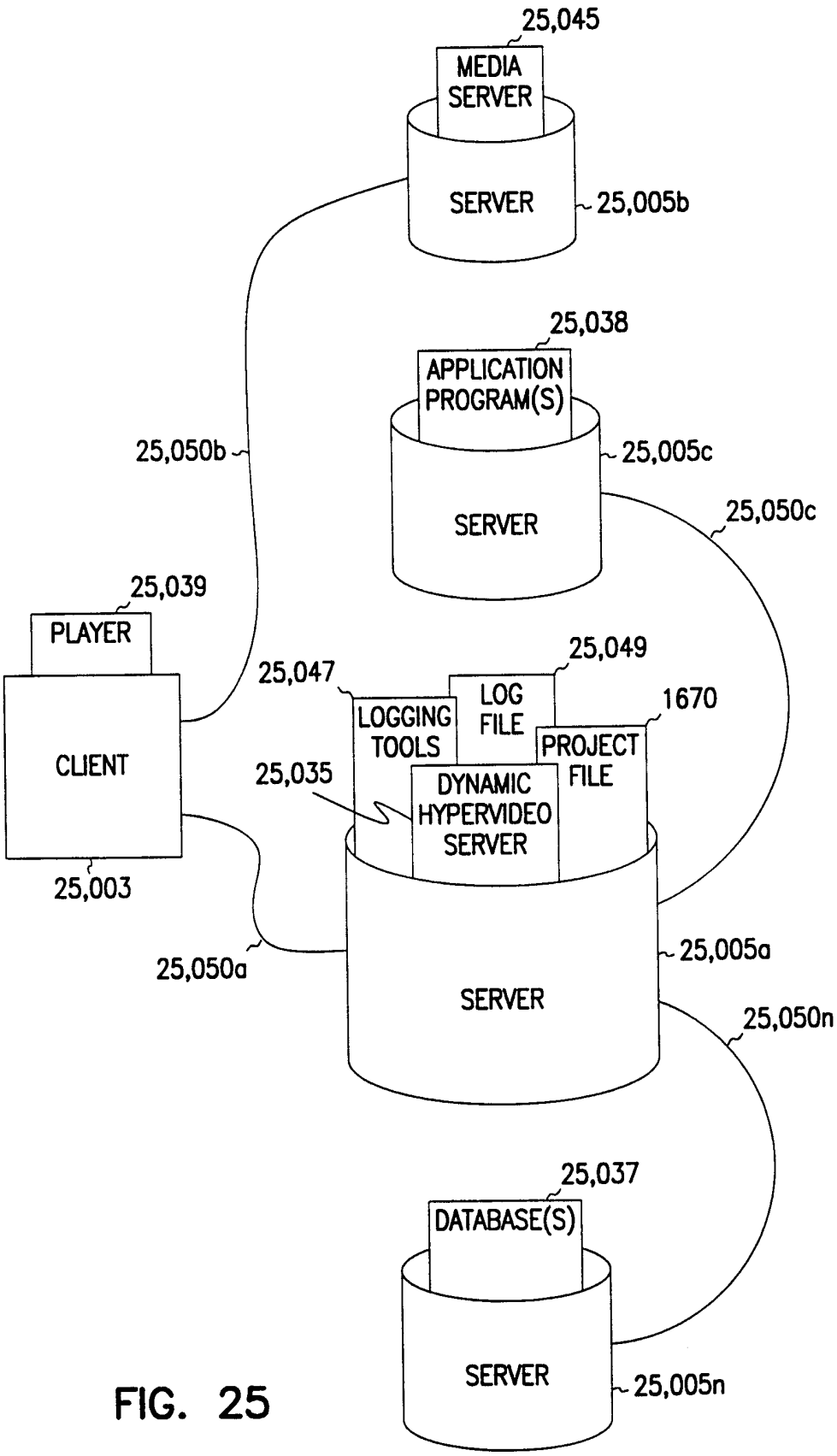

25,045
MEDIA SERVER
SERVER
25,005b
25,038
APPLICATION PROGRAM(S)
25,005c
SERVER
25,050b
25,050c
25,039
PLAYER
CLIENT
25,003
25,047
25,049
LOG FILE
LOGGING TOOLS
1670
PROJECT FILE
25,035
DYNAMIC HYPERVIDEO SERVER
25,050a
25,005a
SERVER
25,050n
25,037
DATABASE(S)
SERVER
25,005n

FIG. 25

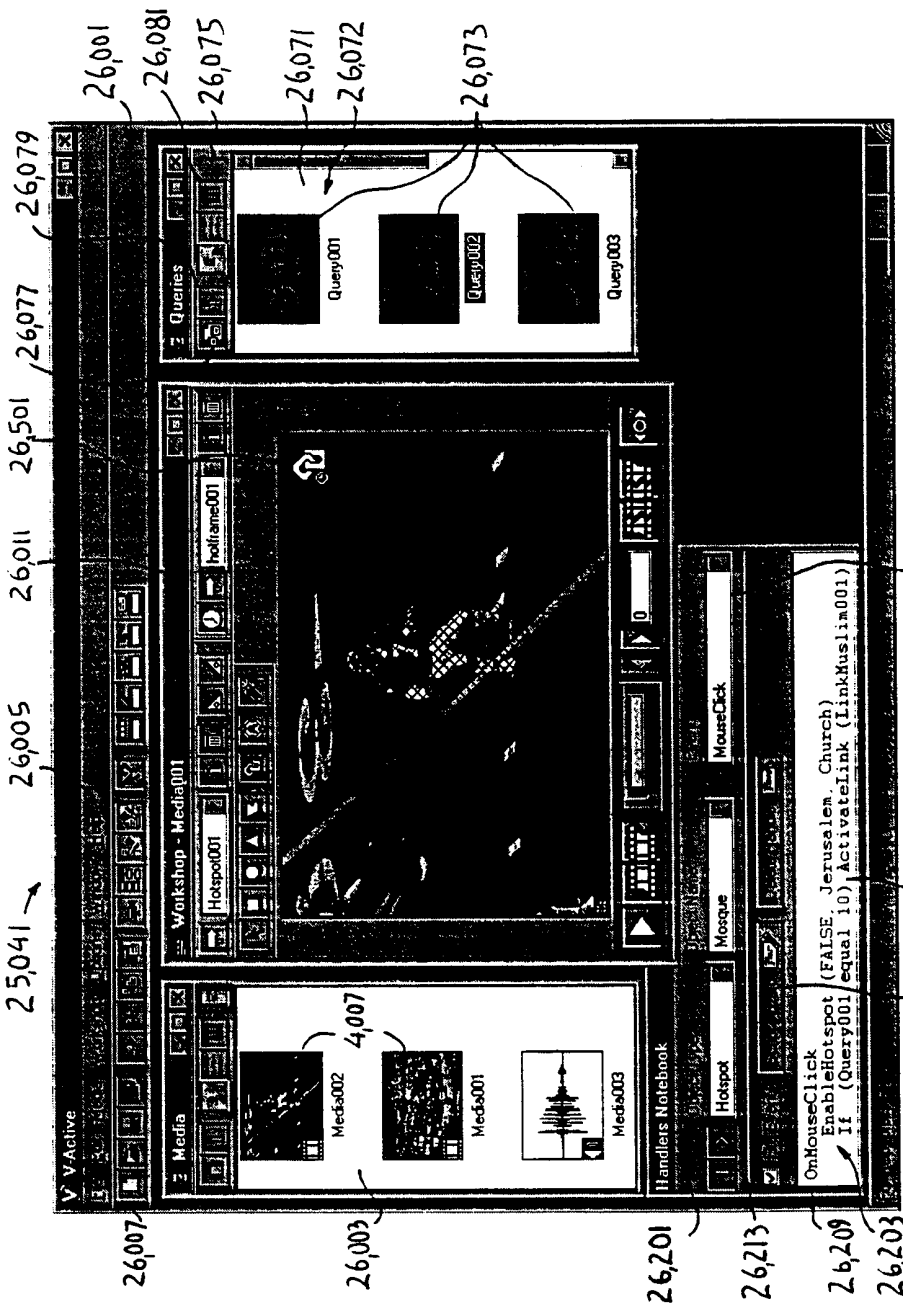

25,041
26,005
26,011
26,501
26,077
26,079
26,001
26,081
26,075
26,071
26,072
26,073
26,007
26,003
4,007
26,201
26,213
26,209
26,203
26,217
26,211
26,205
V Active
Media
Media002
Media001
Media003
Workshop - Media001
Hotspot001
holframe001
Queries
Query001
Query002
Query003
Handlers Notebook
Hotspot
Mosque
MouseClick
OnMouseClick
EnableHotspot (FALSE, Jerusalem, Church)
If (Query001 equal 10) ActivateLink (LinkMuslim001)

FIG. 26

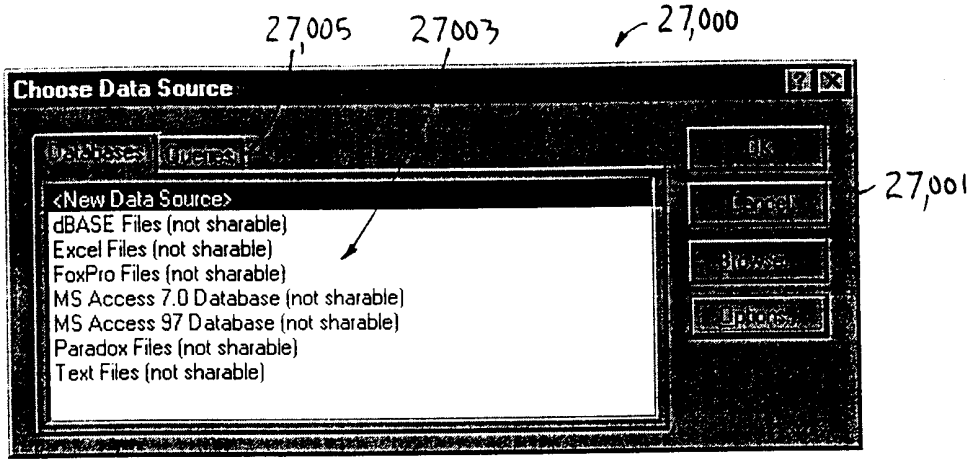

27,005
27,003
27,000
Choose Data Source
Database
Drivers
<New Data Source>
dBASE Files (not sharable)
Excel Files (not sharable)
FoxPro Files (not sharable)
MS Access 7.0 Database (not sharable)
MS Access 97 Database (not sharable)
Paradox Files (not sharable)
Text Files (not sharable)
OK
Cancel
Browse...
Options...
27,001

FIG. 27

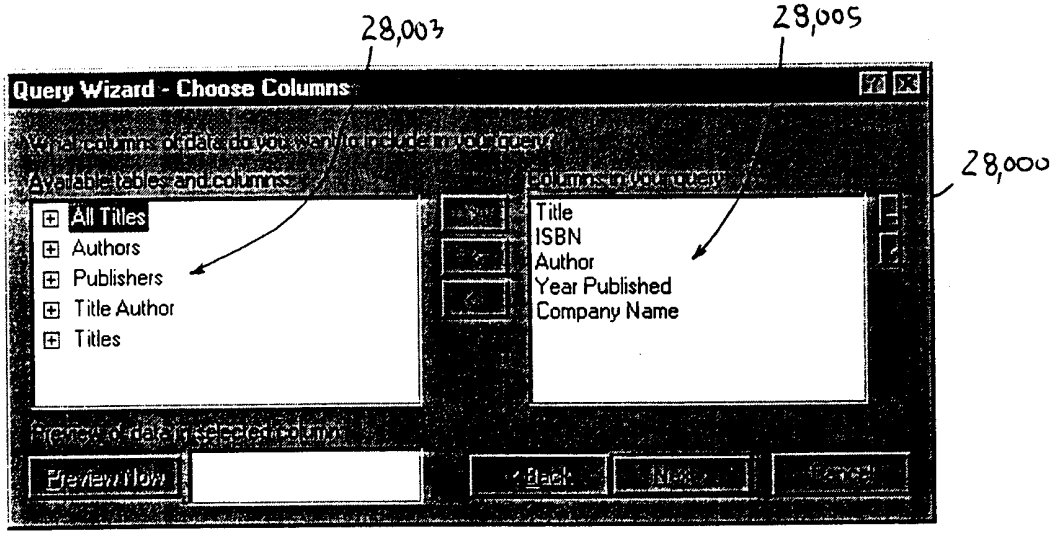

28,003
28,005
28,000
Query Wizard - Choose Columns
Available tables and columns:
Columns in your query:
All Titles
Authors
Publishers
Title Author
Titles
Title
ISBN
Author
Year Published
Company Name
Preview Now
< Back
Next >
Cancel

FIG. 28

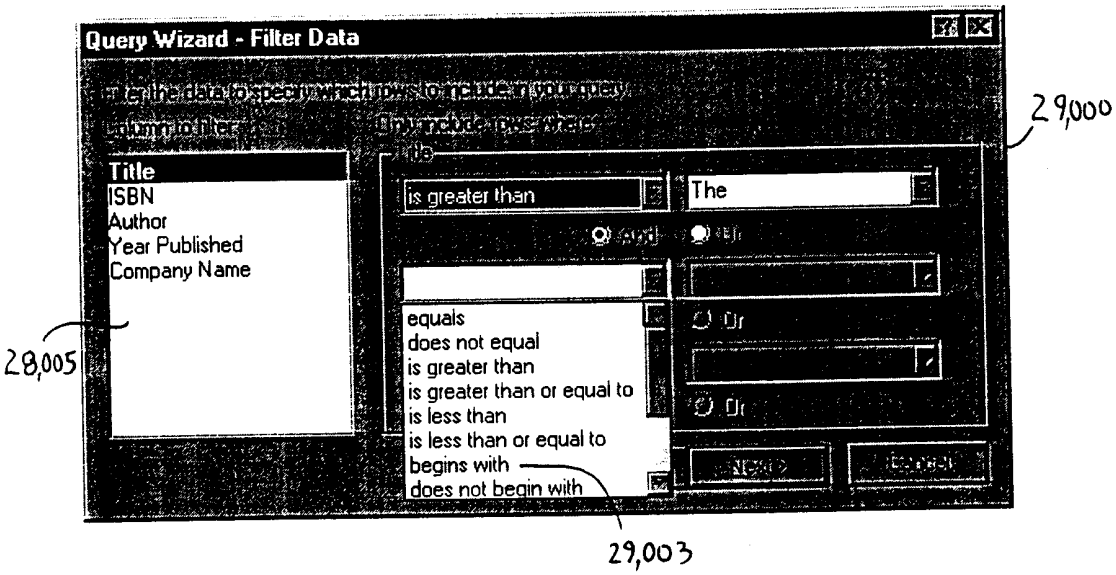

Query Wizard - Filter Data
Title
ISBN
Author
Year Published
Company Name
is greater than
The
equals
does not equal
is greater than
is greater than or equal to
is less than
is less than or equal to
begins with
does not begin with
Or
Or
29,000
28,005
29,003

FIG. 29

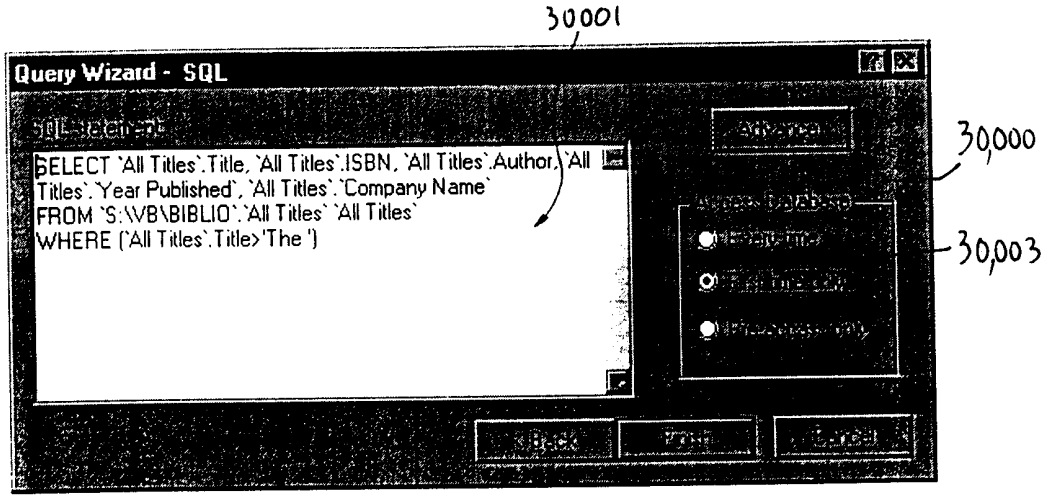

30001
Query Wizard - SQL
SELECT `All Titles`.Title, `All Titles`.ISBN, `All Titles`.Author, `All Titles`.`Year Published`, `All Titles`.`Company Name`
FROM `S:\VB\BIBLIO`.`All Titles` `All Titles`
WHERE (`All Titles`.Title>'The ')
30,000
30,003

FIG. 30

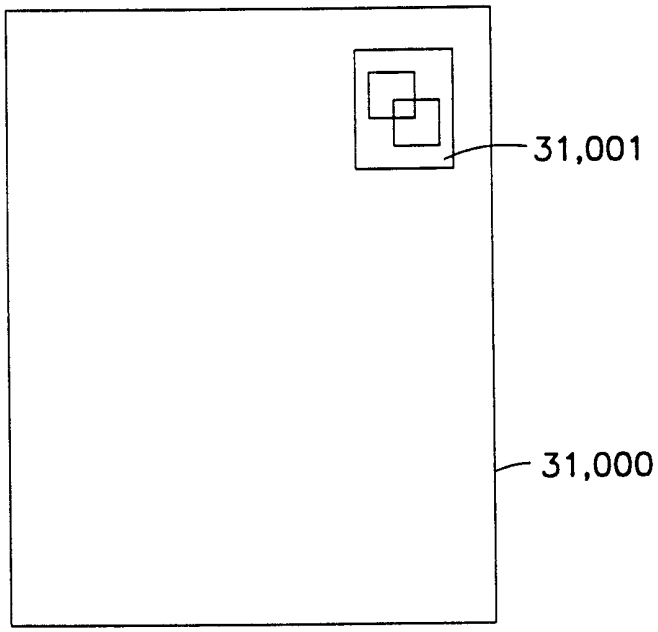


FIG. 31

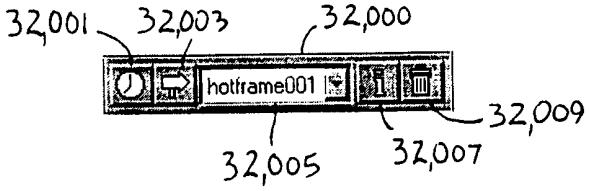


FIG. 32

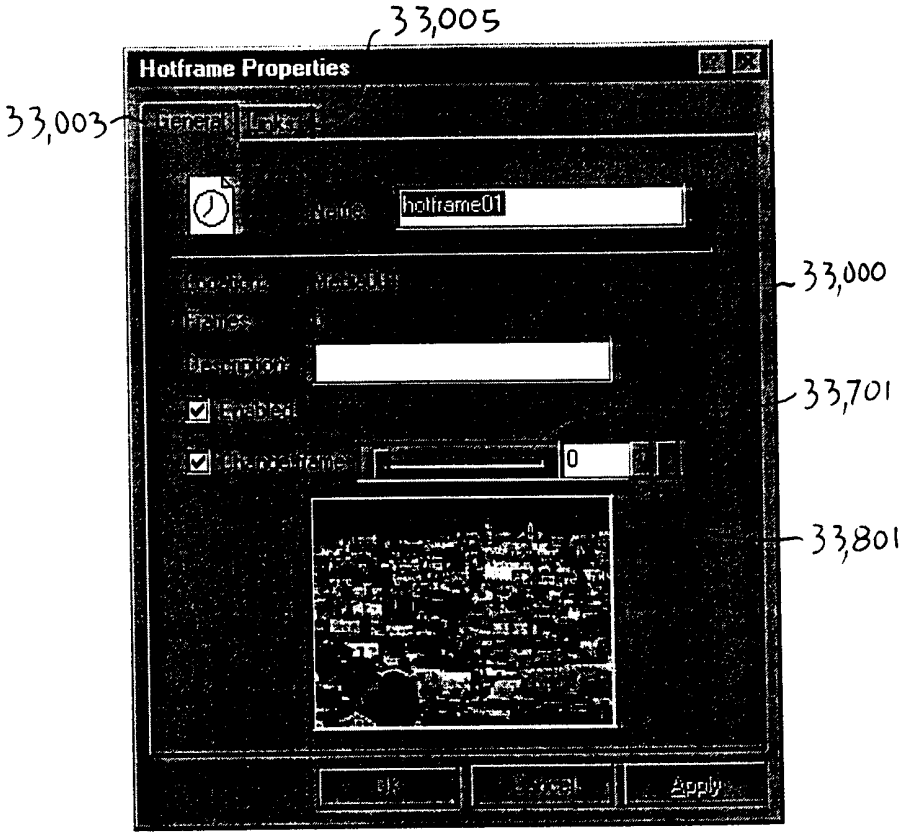

33,005
Hotframe Properties
33,003
General
Locks
hotframe01
33,000
33,701
0
33,801
OK
Cancel
Apply

FIG. 33

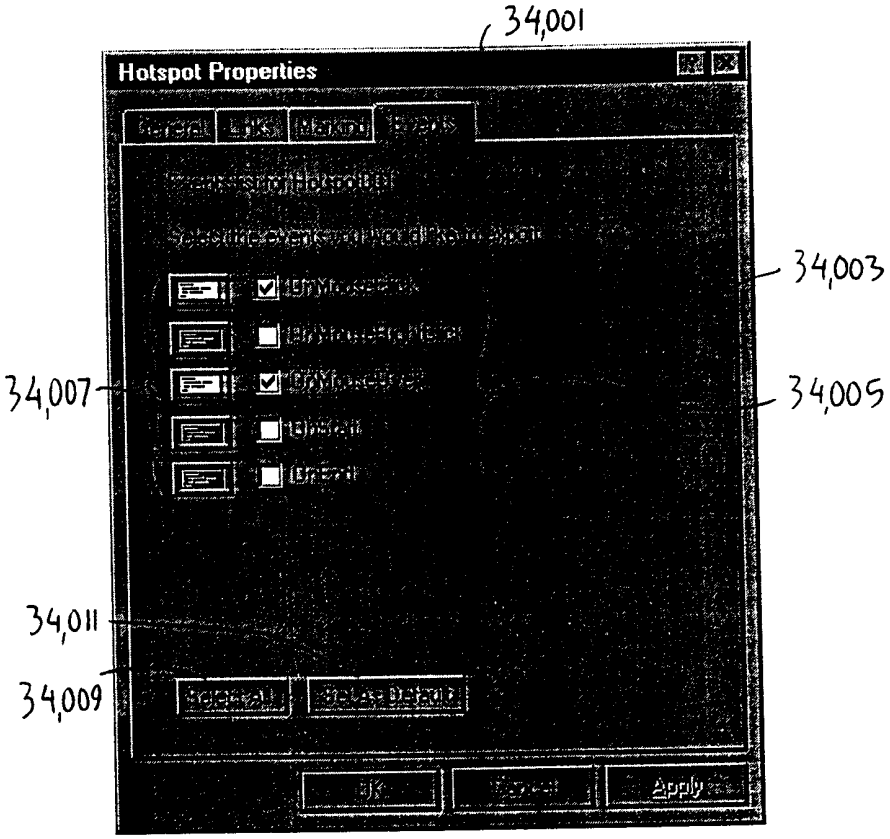

34,001
Hotspot Properties
34,003
34,007
34,005
34,011
34,009
OK
Apply

FIG. 34

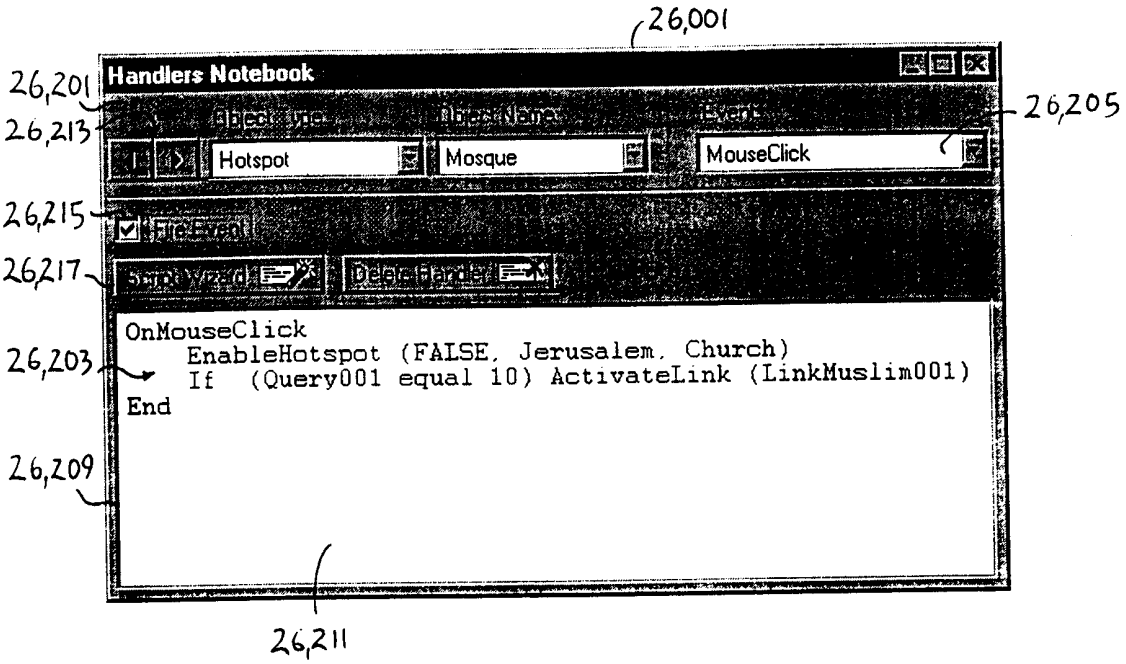

26,001
Handlers Notebook
26,201
26,213
Object Type
Object Name
Event
26,205
Hotspot
Mosque
MouseClick
26,215
Fire Event
26,217
Script Wizard
Delete Handler
OnMouseClick
EnableHotspot (FALSE, Jerusalem, Church)
26,203
If (Query001 equal 10) ActivateLink (LinkMuslim001)
End
26,209
26,211

FIG. 35

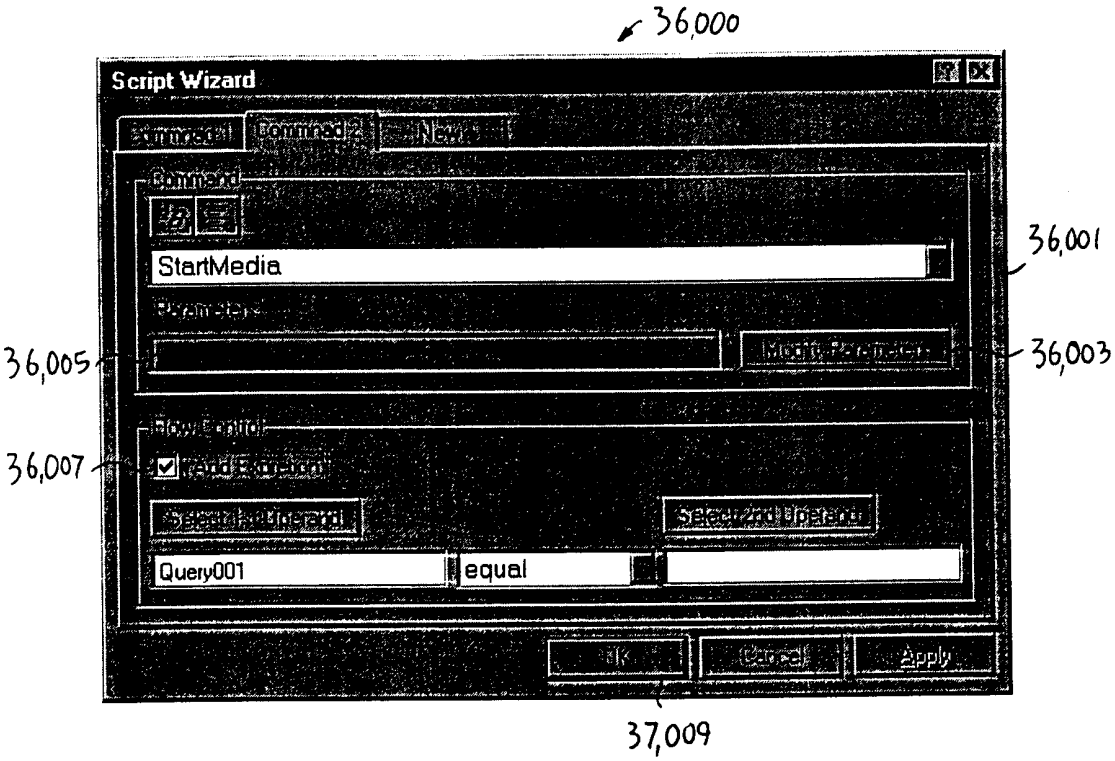

36,000
Script Wizard
StartMedia
36,001
36,005
36,003
36,007
Query001
equal
OK
Cancel
Apply
37,009

FIG. 36

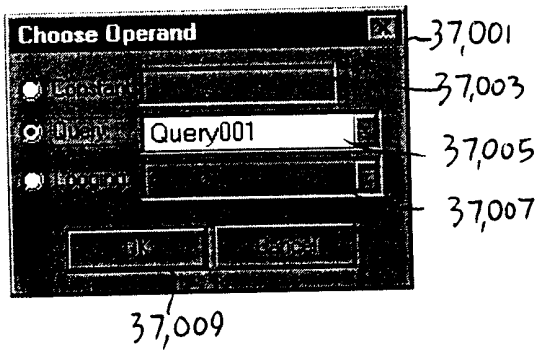


FIG. 37

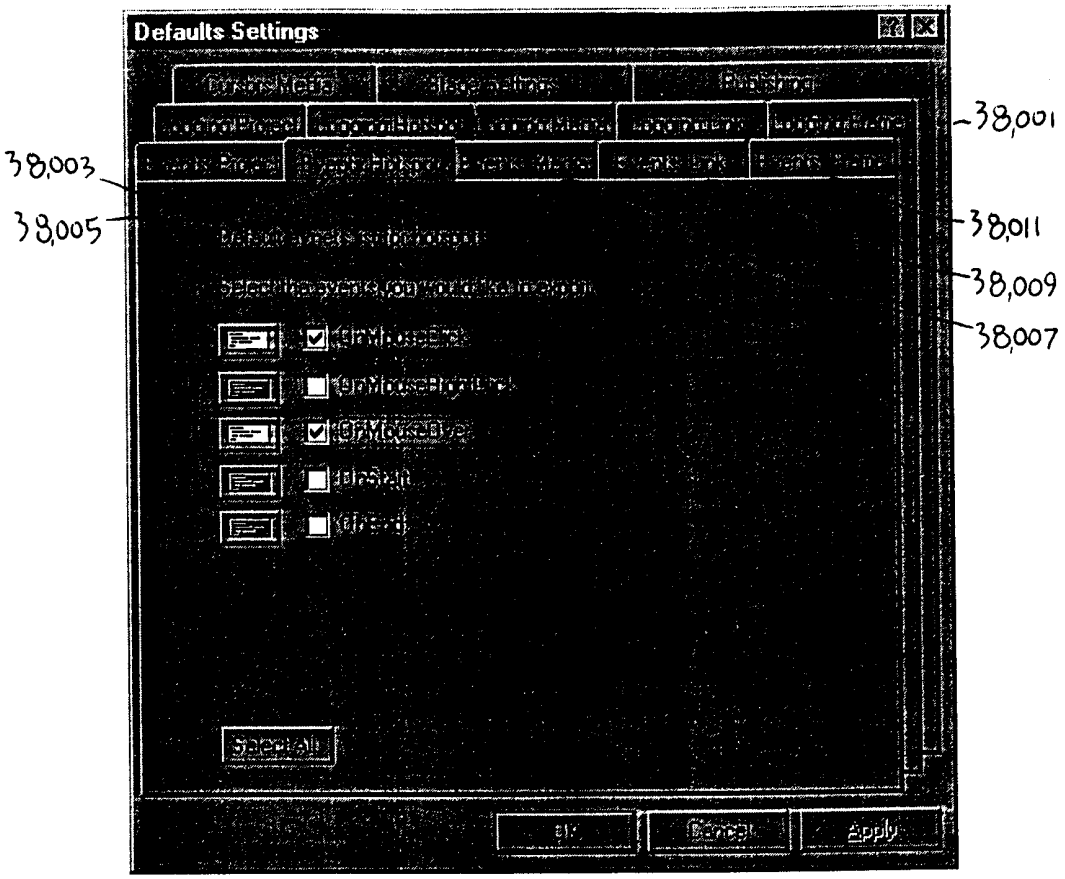

Defaults Settings
38,001
38,003
38,005
38,011
38,009
38,007
OK
Cancel
Apply

FIG. 38

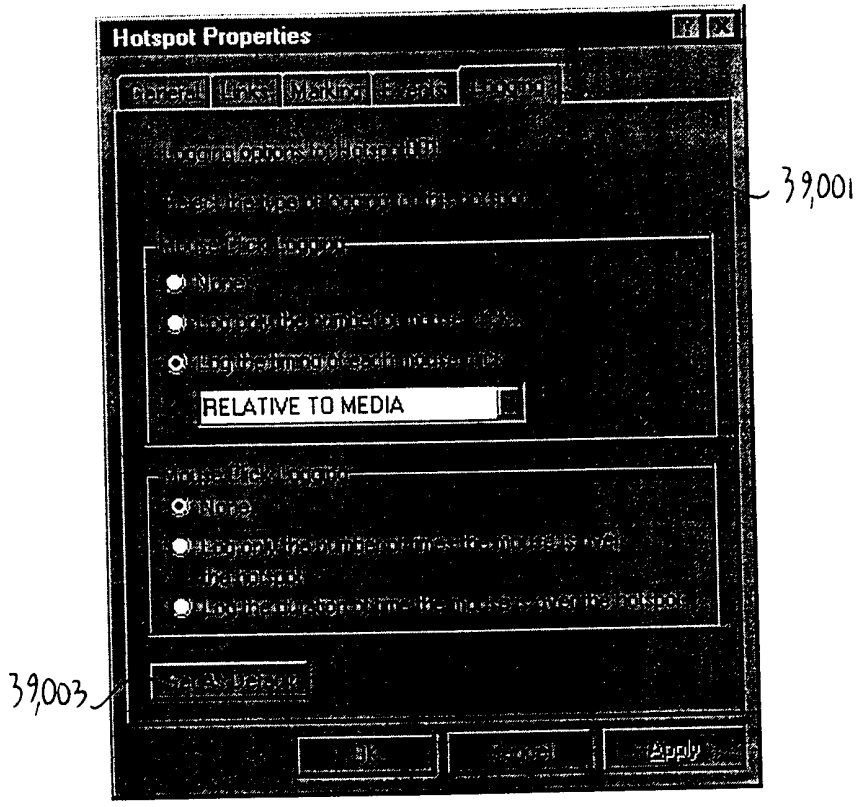


FIG. 39

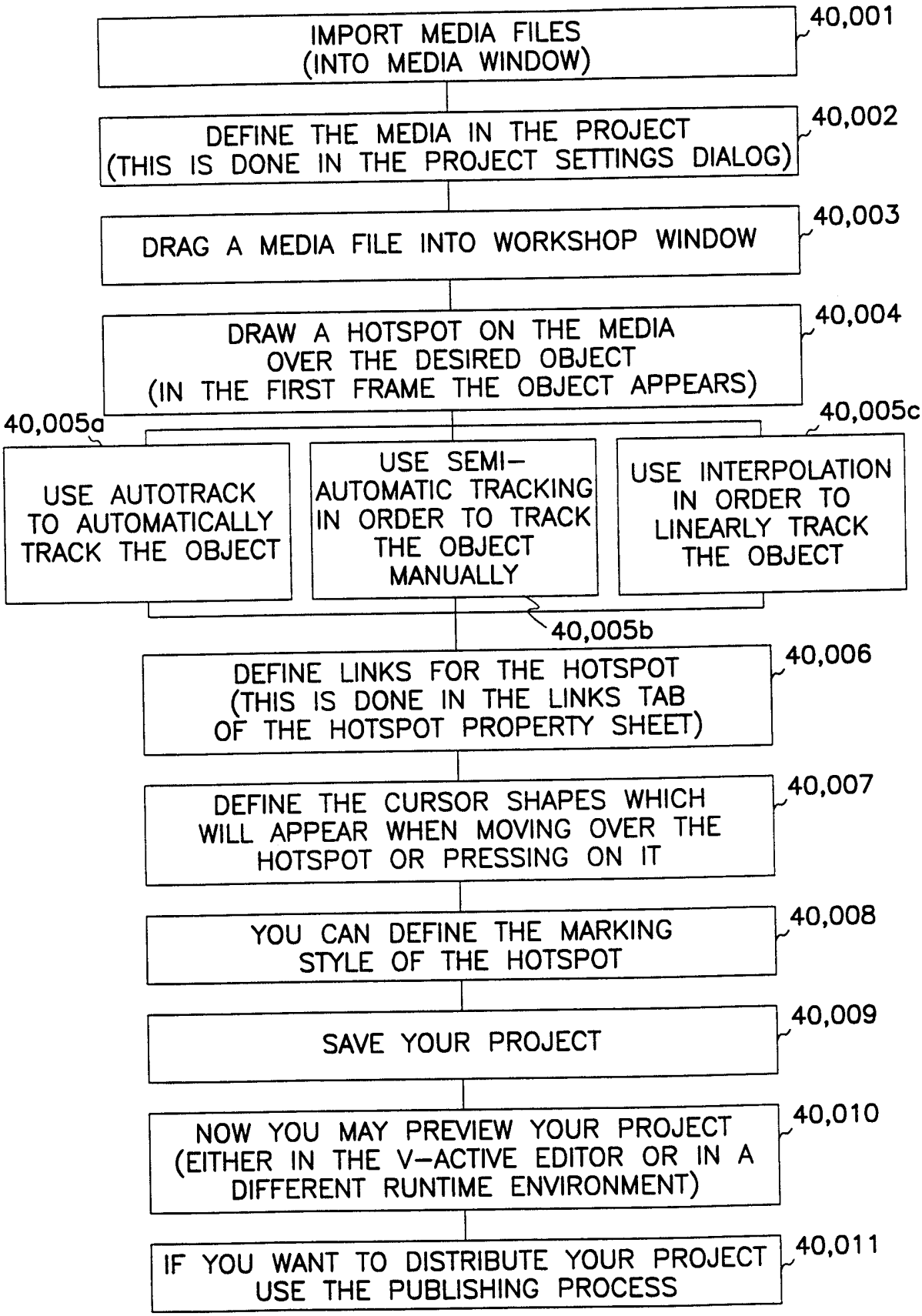

IMPORT MEDIA FILES (INTO MEDIA WINDOW)
40,001
DEFINE THE MEDIA IN THE PROJECT (THIS IS DONE IN THE PROJECT SETTINGS DIALOG)
40,002
DRAG A MEDIA FILE INTO WORKSHOP WINDOW
40,003
DRAW A HOTSPOT ON THE MEDIA OVER THE DESIRED OBJECT (IN THE FIRST FRAME THE OBJECT APPEARS)
40,004
40,005a
USE AUTOTRACK TO AUTOMATICALLY TRACK THE OBJECT
USE SEMI-AUTOMATIC TRACKING IN ORDER TO TRACK THE OBJECT MANUALLY
USE INTERPOLATION IN ORDER TO LINEARLY TRACK THE OBJECT
40,005c
40,005b
DEFINE LINKS FOR THE HOTSPOT (THIS IS DONE IN THE LINKS TAB OF THE HOTSPOT PROPERTY SHEET)
40,006
DEFINE THE CURSOR SHAPES WHICH WILL APPEAR WHEN MOVING OVER THE HOTSPOT OR PRESSING ON IT
40,007
YOU CAN DEFINE THE MARKING STYLE OF THE HOTSPOT
40,008
SAVE YOUR PROJECT
40,009
NOW YOU MAY PREVIEW YOUR PROJECT (EITHER IN THE V-ACTIVE EDITOR OR IN A DIFFERENT RUNTIME ENVIRONMENT)
40,010
IF YOU WANT TO DISTRIBUTE YOUR PROJECT USE THE PUBLISHING PROCESS
40,011

FIG. 40

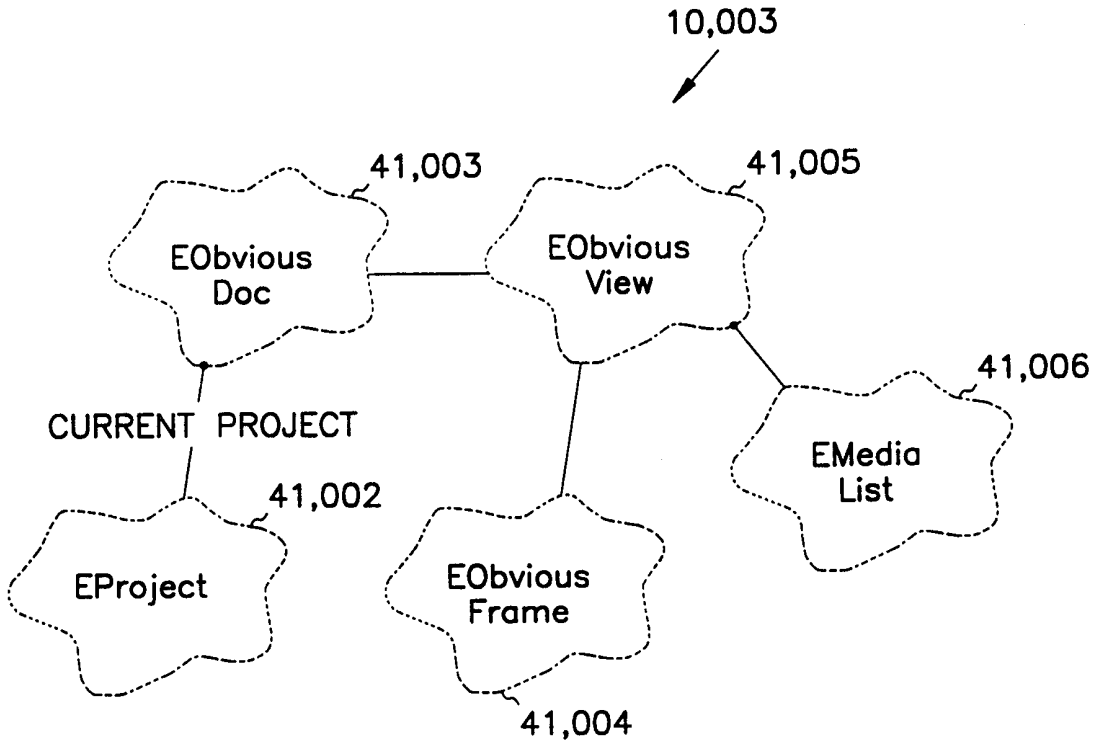

10,003
41,003
41,005
EObvious Doc
EObvious View
41,006
CURRENT PROJECT
EMedia List
41,002
EProject
EObvious Frame
41,004

FIG. 41

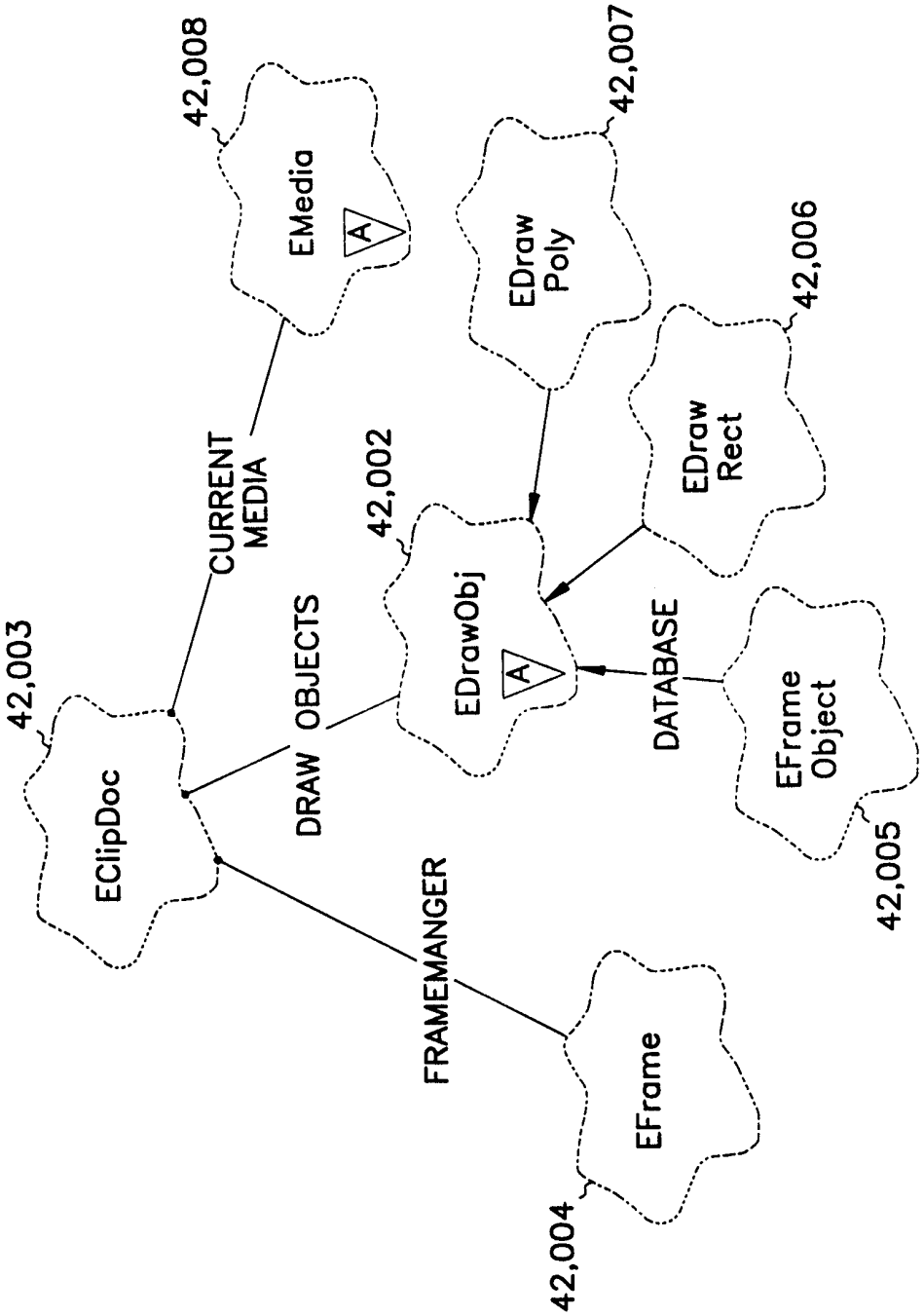

42,008
EMedia
A
42,007
EDraw Poly
42,006
EDraw Rect
CURRENT MEDIA
42,002
EDrawObj
A
42,003
EClipDoc
DRAW OBJECTS
DATABASE
EFrame Object
42,005
FRAMEMANGER
EFrame
42,004

FIG. 42

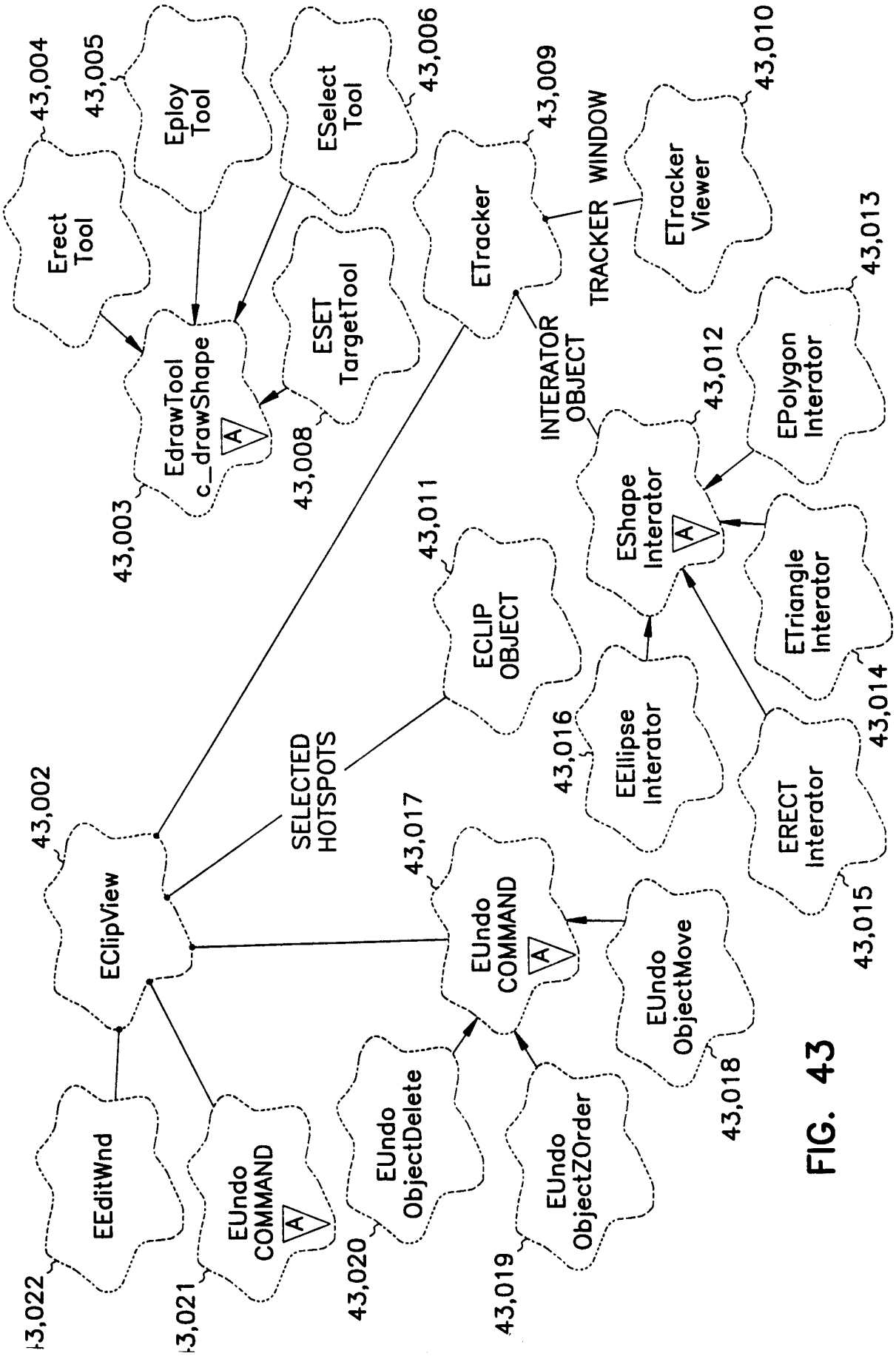

43,004
43,005
Erect Tool
Eploy Tool
ESelect Tool
43,006
43,009
ETracker
TRACKER WINDOW
ETracker Viewer
43,010
EdrawTool c_drawShape
A
43,003
ESET TargetTool
43,008
INTERATOR OBJECT
43,012
EShape Interator
A
EPolygon Interator
43,013
43,011
ECLIP OBJECT
SELECTED HOTSPOTS
ETriangle Interator
43,014
43,016
EEllipse Interator
43,002
EClipView
43,017
EUndo COMMAND
A
EUndo ObjectMove
43,018
ERECT Interator
43,015
EEditWnd
EUndo COMMAND
A
EUndo ObjectDelete
43,020
EUndo ObjectZOrder
43,019
43,022
43,021

FIG. 43

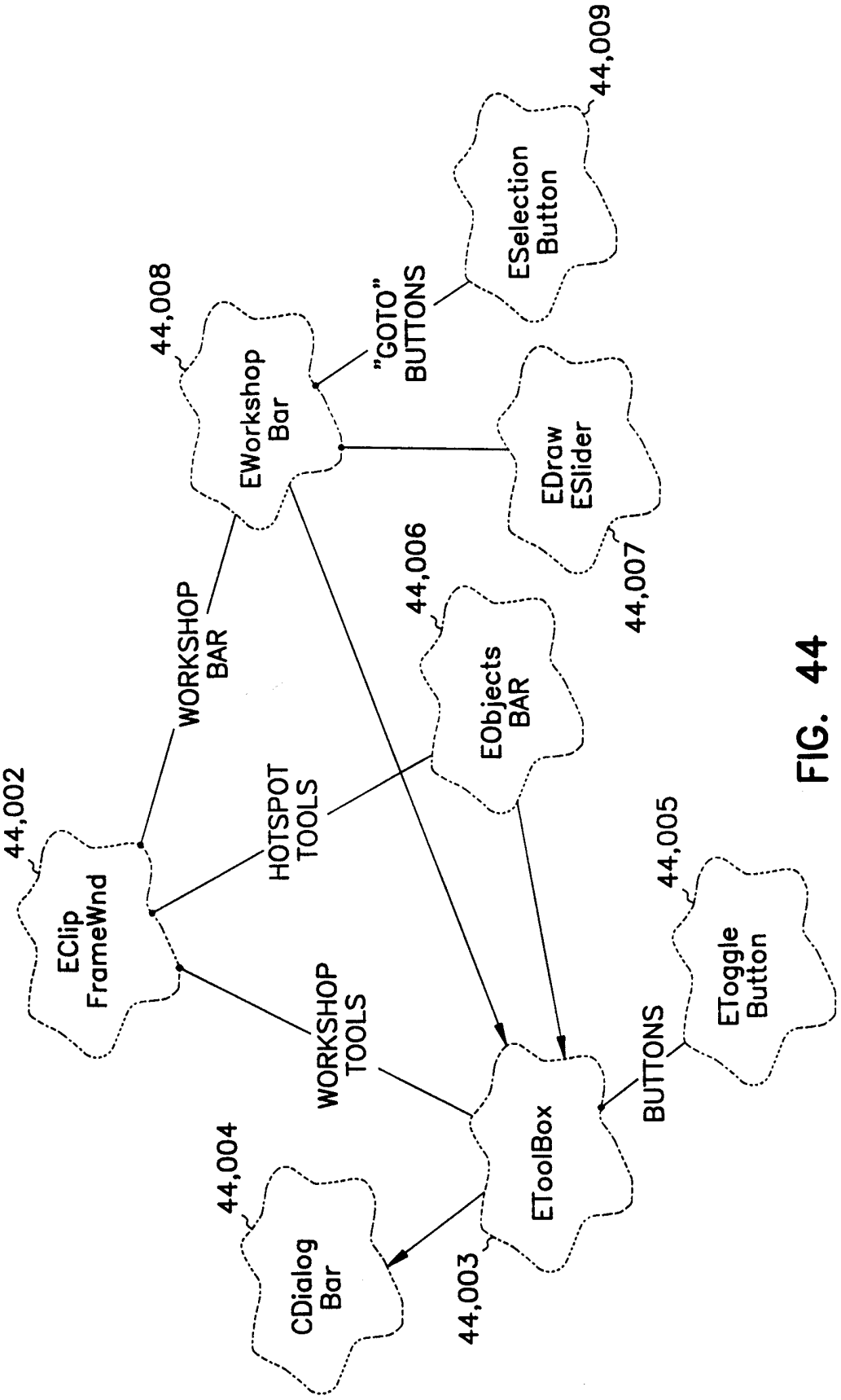

44,009
ESelection Button
"GOTO" BUTTONS
44,008
EWorkshop Bar
EDraw ESlider
44,007
WORKSHOP BAR
44,006
EObjects BAR
HOTSPOT TOOLS
44,002
EClip FrameWnd
WORKSHOP TOOLS
44,005
EToggle Button
BUTTONS
44,004
CDialog Bar
EToolBox
44,003

FIG. 44

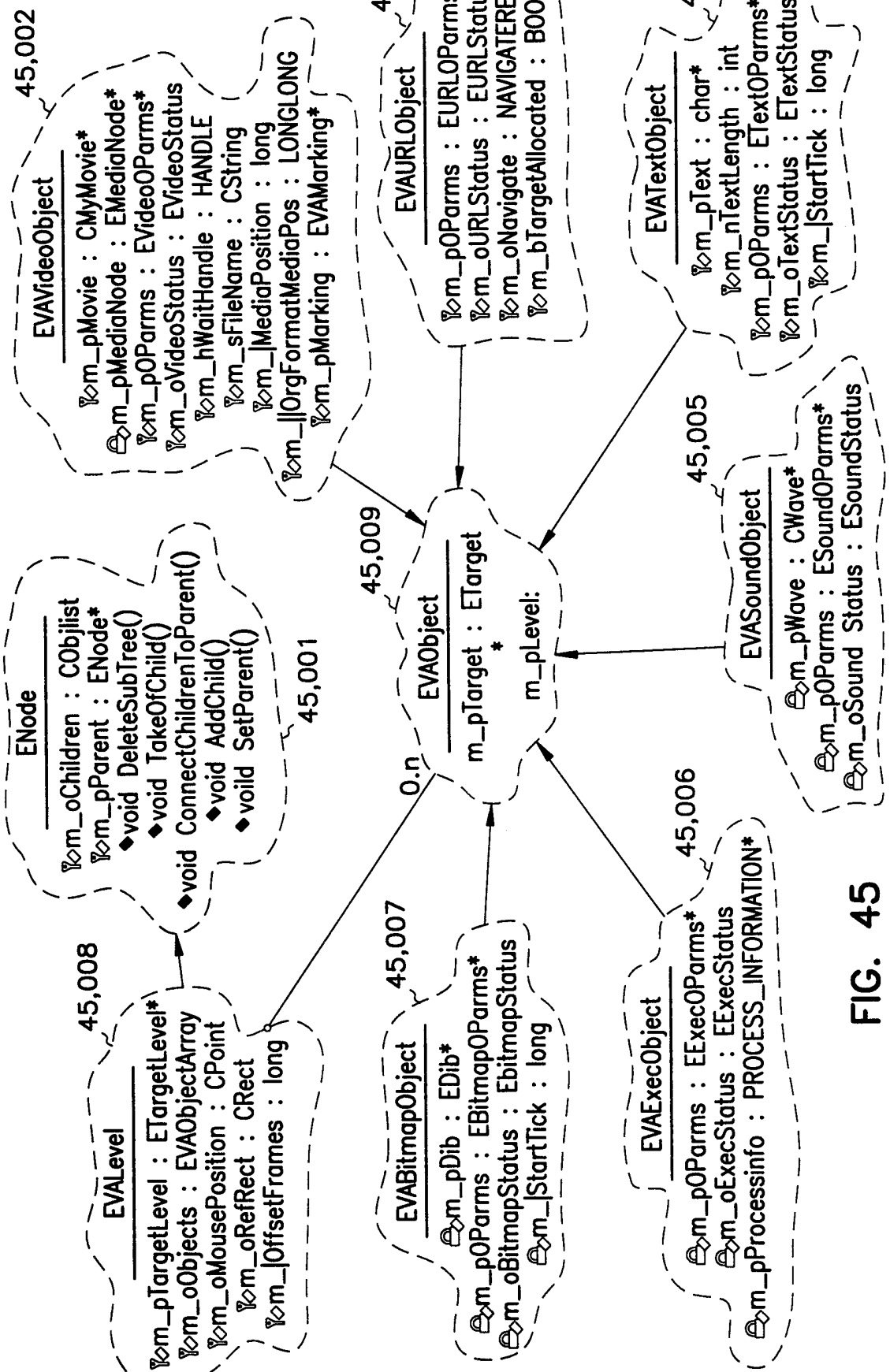

45,002
EVAVideoObject
m_pMovie : CMyMovie*
m_pMediaNode : EMediaNode*
m_pOParms : EVideoOParms*
m_oVideoStatus : EVideoStatus
m_hWaitHandle : HANDLE
m_sFileName : CString
m_lMediaPosition : long
m_lOrgFormatMediaPos : LONGLONG
m_pMarking : EVAMarking*
45,003
EVAURLObject
m_pOParms : EURLOParms*
m_oURLStatus : EURLStatus
m_oNavigate : NAVIGATEREC
m_bTargetAllocated : BOOL
45,004
EVATextObject
m_pText : char*
m_nTextLength : int
m_pOParms : ETextOParms*
m_oTextStatus : ETextStatus
m_lStartTick : long
45,005
EVASoundObject
m_pWave : CWave*
m_pOParms : ESoundOParms*
m_oSound Status : ESoundStatus
45,006
EVAExecObject
m_pOParms : EExecOParms*
m_oExecStatus : EExecStatus
m_pProcessinfo : PROCESS_INFORMATION*
45,007
EVABitmapObject
m_pDib : EDib*
m_pOParms : EBitmapOParms*
m_oBitmapStatus : EbitmapStatus
m_lStartTick : long
45,008
EVALevel
m_pTargetLevel : ETargetLevel*
m_oObjects : EVAObjectArray
m_oMousePosition : CPoint
m_oRefRect : CRect
m_lOffsetFrames : long
45,009
EVAObject
m_pTarget : ETarget
*
m_pLevel:
0..n
45,001
ENode
m_oChildren : CObjlist
m_pParent : ENode*
void DeleteSubTree()
void TakeOfChild()
void ConnectChildrenToParent()
void AddChild()
voild SetParent()

FIG. 45

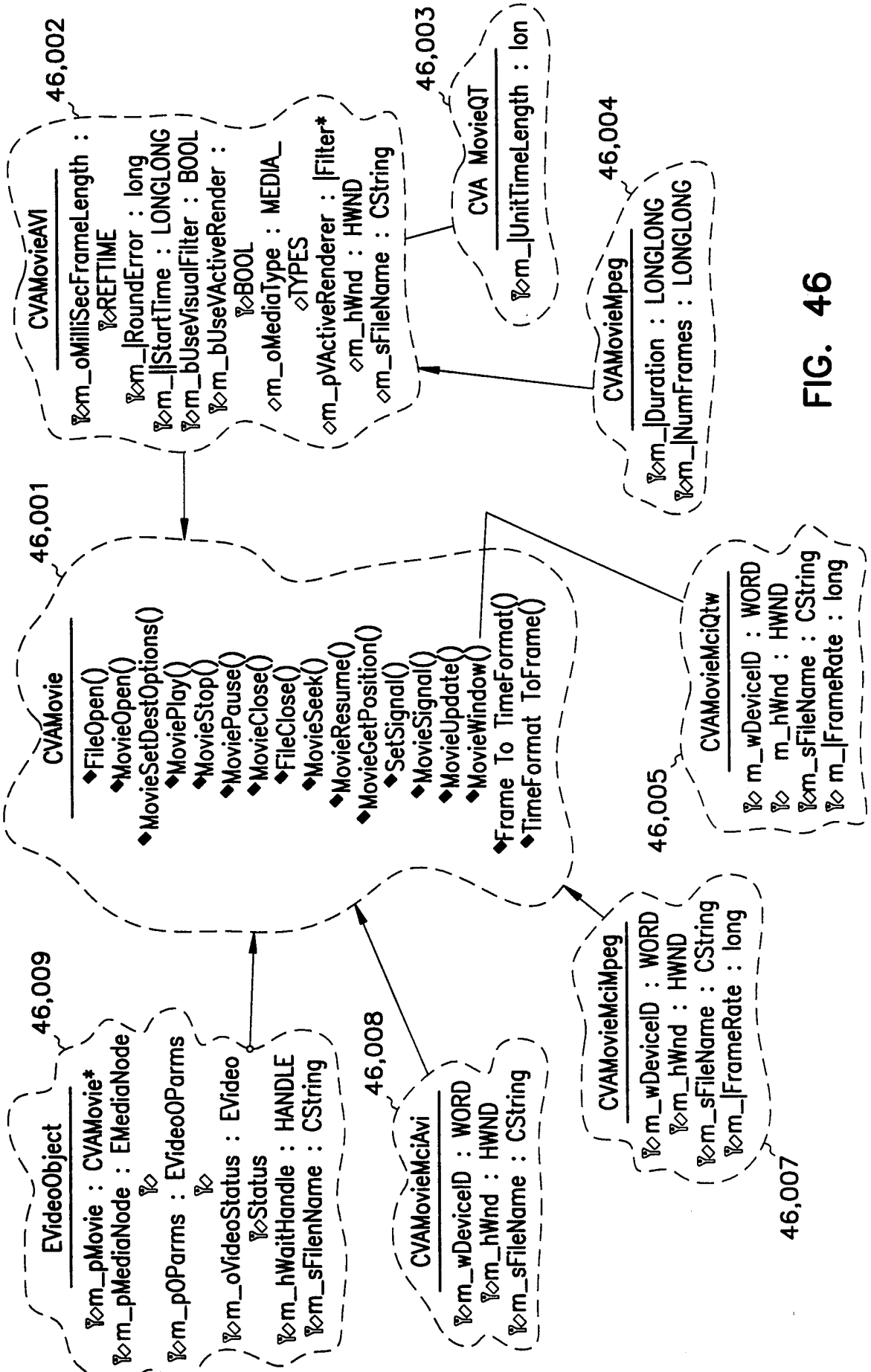

46,002
CVAMovieAVI
m_oMilliSecFrameLength : REFTIME
m_lRoundError : long
m_lStartTime : LONGLONG
m_bUseVisualFilter : BOOL
m_bUseVActiveRender : BOOL
m_oMediaType : MEDIA_TYPES
m_pVActiveRenderer : IFilter*
m_hWnd : HWND
m_sFileName : CString
46,003
CVA MovieQT
m_lUnitTimeLength : lon
46,004
CVAMovieMpeg
m_lDuration : LONGLONG
m_lNumFrames : LONGLONG
46,001
CVAMovie
FileOpen()
MovieOpen()
MovieSetDestOptions()
MoviePlay()
MovieStop()
MoviePause()
MovieClose()
FileClose()
MovieSeek()
MovieResume()
MovieGetPosition()
SetSignal()
MovieSignal()
MovieUpdate()
MovieWindow()
Frame To TimeFormat()
TimeFormat ToFrame()
46,005
CVAMovieMciQtw
m_wDeviceID : WORD
m_hWnd : HWND
m_sFileName : CString
m_lFrameRate : long
46,009
EVideoObject
m_pMovie : CVAMovie*
m_pMediaNode : EMediaNode
m_pOParms : EVideoOParms
m_oVideoStatus : EVideo Status
m_hWaitHandle : HANDLE
m_sFilenName : CString
46,008
CVAMovieMciAvi
m_wDeviceID : WORD
m_hWnd : HWND
m_sFileName : CString
CVAMovieMciMpeg
m_wDeviceID : WORD
m_hWnd : HWND
m_sFileName : CString
m_lFrameRate : long
46,007

FIG. 46

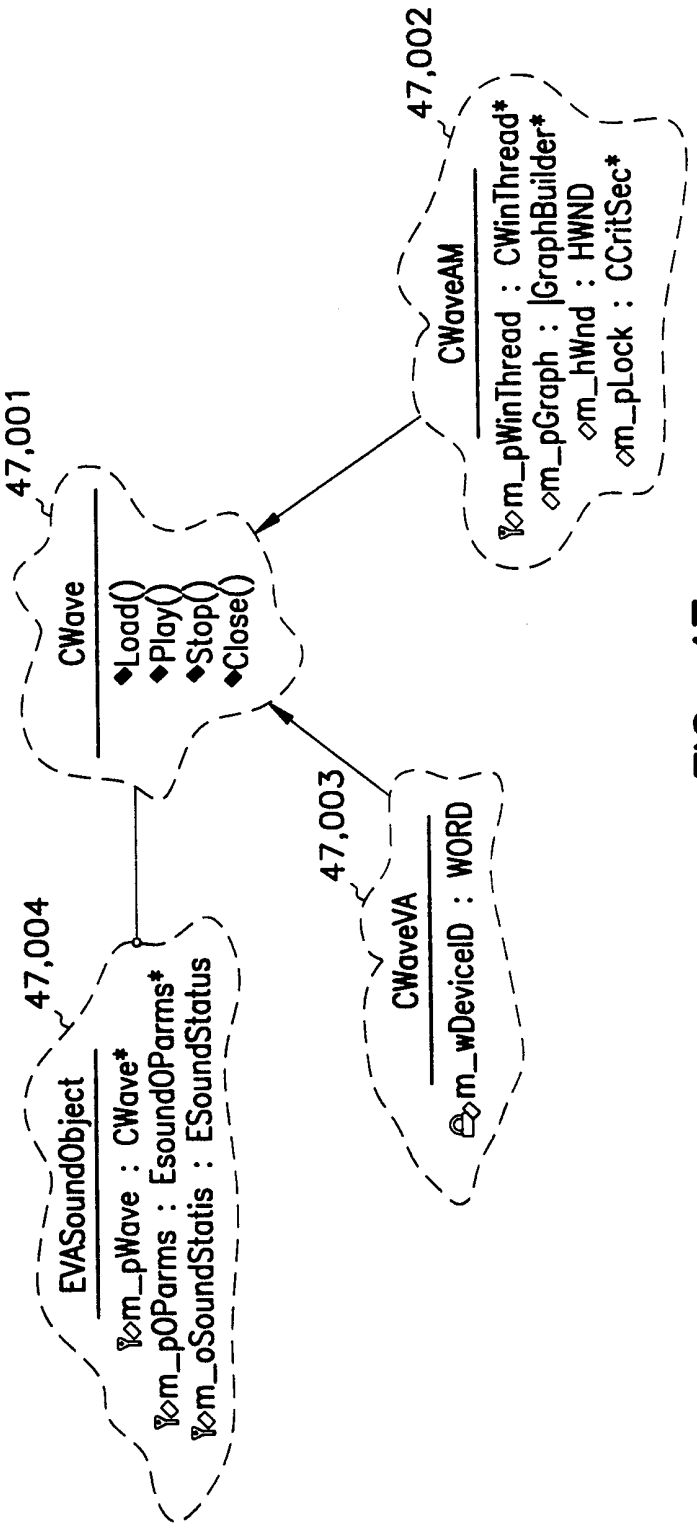

47,001
CWave
◆Load()
◆Play()
◆Stop()
◆Close()
47,002
CWaveAM
◇m_pWinThread : CWinThread*
◇m_pGraph : IGraphBuilder*
◇m_hWnd : HWND
◇m_pLock : CCritSec*
47,003
CWaveVA
◇m_wDeviceID : WORD
47,004
EVASoundObject
◇m_pWave : CWave*
◇m_pOParms : EsoundOParms*
◇m_oSoundStatis : ESoundStatus

FIG. 47

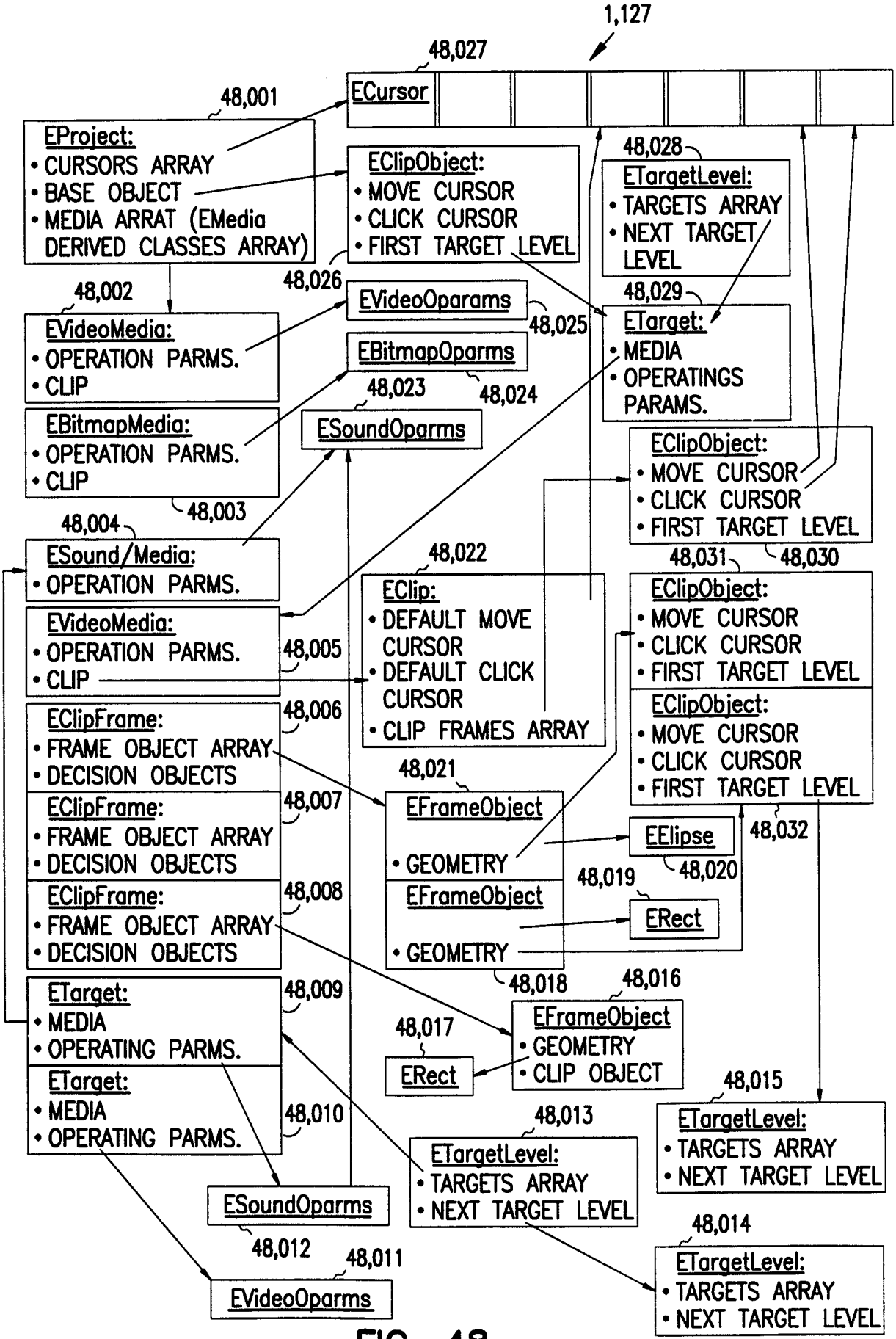

1,127
48,027
ECursor
48,001
EProject:
• CURSORS ARRAY
• BASE OBJECT
• MEDIA ARRAT (EMedia DERIVED CLASSES ARRAY)
EClipObject:
• MOVE CURSOR
• CLICK CURSOR
• FIRST TARGET LEVEL
48,028
ETargetLevel:
• TARGETS ARRAY
• NEXT TARGET LEVEL
48,026
48,002
EVideoMedia:
• OPERATION PARMS.
• CLIP
EVideoOparams
48,025
48,029
ETarget:
• MEDIA
• OPERATINGS PARAMS.
EBitmapOparms
48,024
48,023
ESoundOparms
EBitmapMedia:
• OPERATION PARMS.
• CLIP
48,003
EClipObject:
• MOVE CURSOR
• CLICK CURSOR
• FIRST TARGET LEVEL
48,004
ESound/Media:
• OPERATION PARMS.
48,022
48,031
48,030
EClip:
• DEFAULT MOVE CURSOR
• DEFAULT CLICK CURSOR
• CLIP FRAMES ARRAY
EClipObject:
• MOVE CURSOR
• CLICK CURSOR
• FIRST TARGET LEVEL
EVideoMedia:
• OPERATION PARMS.
• CLIP
48,005
48,006
EClipFrame:
• FRAME OBJECT ARRAY
• DECISION OBJECTS
48,021
EClipObject:
• MOVE CURSOR
• CLICK CURSOR
• FIRST TARGET LEVEL
48,007
EClipFrame:
• FRAME OBJECT ARRAY
• DECISION OBJECTS
EFrameObject
• GEOMETRY
EElipse
48,020
48,032
48,019
ERect
48,008
EClipFrame:
• FRAME OBJECT ARRAY
• DECISION OBJECTS
EFrameObject
• GEOMETRY
48,018
48,016
48,009
ETarget:
• MEDIA
• OPERATING PARMS.
EFrameObject
• GEOMETRY
• CLIP OBJECT
48,017
ERect
48,015
ETarget:
• MEDIA
• OPERATING PARMS.
48,010
48,013
ETargetLevel:
• TARGETS ARRAY
• NEXT TARGET LEVEL
ETargetLevel:
• TARGETS ARRAY
• NEXT TARGET LEVEL
ESoundOparms
48,012
48,014
48,011
EVideoOparms
ETargetLevel:
• TARGETS ARRAY
• NEXT TARGET LEVEL

FIG. 48

INTERNATIONAL SEARCH REPORT

Ir ational Application No
PCT/US 98/17444

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

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

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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	"MULTIMEDIA HYPERVIDEO LINKS FOR FULL MOTION VIDEOS" IBM TECHNICAL DISCLOSURE BULLETIN, vol. 37, no. 4A, 1 April 1994, page 95 XP000446196 see the whole document ---	1-11
X	BURRILL V ET AL: "TIME-VARYING SENSITIVE REGIONS IN DYNAMIC MULTIMEDIA OBJECTS: A PRAGMATIC APPROACH TO CONTENT BASED RETRIEVAL FROM VIDEO" INFORMATION AND SOFTWARE TECHNOLOGY, vol. 36, no. 4, 1 January 1994, pages 213-223, XP000572844 see the whole document --- -/--	1-11

[X] Further documents are listed in the continuation of box C.

[ ] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
1 December 1998	09/12/1998

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Katerbau, R

1

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

INTERNATIONAL SEARCH REPORT

In ational Application No
PCT/US 98/17444

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	GINIGE A ET AL: "HYPERMEDIA AUTHORING" IEEE MULTIMEDIA, vol. 2, no. 4, 21 December 1995, pages 24-35, XP000542076 see the whole document	12-14, 17,18

1