INFORMATION STORAGE AND RETRIEVAL SYSTEM FOR STORING AND RETRIEVING THE VISUAL FORM OF INFORMATION FROM AN APPLICATION IN A DATABASE

BACKGROUND

Computer programs generally maintain data in a variety of formats. There usually is one format that is unique, and typically proprietary, to each computer program in which raw data are stored persistently. This format usually is designed to reduce the amount of information actually stored and, in some cases, to restrict the ability of a third party to access the data. Data in this format generally are created by a "save" function of the computer program. The save function formats the raw data and stores the formatted raw data in yet another format, called a "file," that is defined by the operating system for which the computer program is designed. Data that are being processed by a computer program are stored in another format, also typically proprietary, called a "data structure," which generally is stored in volatile or working memory during execution of the computer program. A data structure usually is designed to permit the data to be processed efficiently by the computer program, while minimizing the amount of memory needed to represent the data.

With many computer programs, the most useful form of the data from the perspective of the user is its visual form, e.g., what is displayed on a computer display or what is printed. However, this form of the data often is not captured into permanent or persistent storage, unless it is printed and the printed form is electronically scanned. In particular, the file format used by a computer program often does not maintain data in a visual form for several reasons. The visual form of the data generally requires more information to be represented and can be reconstructed from raw data that require less information to be represented. Therefore the need to store the visual form of the data is generally considered unnecessary.

Part of the visual form of data produced by a computer program is generated, for example, from environmental data (such as the date and time) or user selected data that are being processed, and is not recoverable from the file format, but only from the data structures

of the computer program. Although some data structures represent the visual form of the data, often there is no mechanism to retain the visual form of the data other than by printing. Some operating systems permit displayed data to be copied from one computer program to another using a "cut-and-paste" operation, but this operation generally requires the other computer program to be in operation on the same machine. Some computer programs also do not have these operations available to the user. For some computer programs, the printed form of the data, not the displayed data, is most useful and this operation does not provide access to the printed data.

Even if the visual form of data from a computer program were stored, as new versions of the computer program are used, or if the computer program is no longer available, access to that data is impeded. Also, another computer program still might not be able to access the data if the data are stored in a proprietary format.

This lack of access to the visual form of the data from a computer program creates a variety of problems when this form of the data is desired for creating compound documents from multiple sources of data, particularly if the data are created, used and shared over a period of time by multiple different users with multiple different computer programs that are dispersed geographically. As a particular example, in the pharmaceutical industry, data may be acquired from many laboratory instruments in geographically dispersed laboratories over a significant period of time, and then may be combined to produce reports, for example, for regulatory compliance. The inability to centrally access an electronic visual form of the data from these instruments adds a significant cost to regulatory compliance.

Electronically scanning printed documents to provide shared electronic access to such documents has several disadvantages. First, scanning consumes significant resources, including time and effort of personnel. Second, a significant time delay between the creation of a document and its availability to others may occur. Third, bit mapped images created by scanning become distorted when scaled, rotated or otherwise transformed. Fourth, in order for text to be searchable in a scanned document, the scanned document must be processed by optical character recognition (OCR) software.

Another problem that may be encountered with data storage is that data integrity may be compromised, either intentionally or accidentally, between the time when the data are stored and the time when the data are used. If the data are being used to obtain regulatory or administrative approval, some assurance of the integrity of the data may be required.

SUMMARY

The visual form of data may be received from a computer program and may be stored in a database. The visual form of the data may be received, for example, in response to a print operation by the computer program or by some other operation such as a cut and paste sequence of operations or by sending the data to another application. The visual form of the data may be stored as a vector image that permits scalability. The visual form of the data may be stored with other identifying information or tags in the database to facilitate searching of the database. The data in the database may be encoded in a manner that ensures data integrity and that permits detection of when data has been compromised. In one embodiment, a service layer application is provided to control access to the database by performing encoding and decoding of the data in the database. The service layer may have an application programming interface that permits many applications to have access to the database. Another application may be provided for accessing the visual form of the data from the database and for providing this data to another computer program. Such an application permits a user to create compound documents from data in the database using the other computer program.

Accordingly, the following represent some of the aspects of the invention, either alone or in any combination together. The various aspects of the invention may be as a computer-implemented process, computer system or a computer program product with computer-readable code encoded on a tangible information recording or transmission medium. Yet further aspects of the invention will be apparent from the following detailed description of an example implementation, and modifications thereto.

In one aspect, the visual form of information is captured from an application executed on a computer into a database. The visual form of the data is received from the application. Tags corresponding to the visual form of the data also are received. The visual form of the data as a vector image and the tags are stored into the database.

In another aspect, the database which stores the visual form of data from an application executed on a computer may include a first table for storing tags corresponding to the visual form of the data and a reference to the visual form of the data and a second table for storing the visual form of the data as a vector image with a reference to the visual form of the data.

In another aspect, a database in which visual form of data from applications executed on one or more computers are stored as vector images is accessed. The database may be

searched to identify one or more vector images. The identified vector images are sorted according to tags associated with the identified vector images. A user is permitted to select one or more of the identified vector images, which are then displayed. The displayed vector images may be scaled and the scaled vector image is displayed. Text in the selected vector images also may be searched.

In another aspect, a selected portion of visual form of data may be sent from a first application to a second application. An indication of a selected area of the visual form of the data is received in the first application. An indication of the second application also is received. Keystrokes in the second application for pasting data into the second application are determined. Data representing the selected area of the visual form of the data are determined and transferred to a memory area from which the second application may paste data. The determined keystrokes are sent to the second application.

In another aspect, a visual form of text data is converted into character delimited text data for insertion into tabular structure. An indication of a selected area of the visual form of the data is received. Data representing the presence of text within the selected area along a horizontal extent for all rows of text is initialized. The initialized data is modified to indicate the presence of text within the selected area along the horizontal extent for all rows of text. Character delimited data is generated according to the text in the row and the column boundaries identified in the row.

In another aspect, text in a vector image including commands specifying text and horizontal and vertical coordinates for placement of the text is searched. The commands in the vector image are sorted according to the vertical coordinates, then the horizontal coordinates. A list structure representing an ordering of the text from the ordered commands is generated. The list structure may be a doubly linked list. The list structure may combine text strings close to one another so text can be located even though it may be split across multiple text records. The text in each selected metafile is processed into an ordered list so that words may be searched in the order that they appear within each metafile. Such ordering also permits any previous or next occurrences of a word or other character string to be identified in the metafile and among a set of metafiles. Documents from many different applications may be searched together using this technique.

In another aspect, access to a database is controlled by a service layer. In response to a request to store data in the database, the data is encoded and an error checking calculation is

performed on the data. The encoded data and a result of the error checking calculation are stored in the database. In response to a request to read data from the database, the read data is decoded and an error checking calculation is performed on the read data. The result of the error checking calculation of the read data is compared with the stored result of the error checking calculation performed on the stored data to identify any discrepancy. The decoded read data and an indication of any discrepancy identified between the stored and read data is provided.

In another aspect, a database for stores the visual form of data with corresponding tags identifying the visual form of the data. Each of the plurality of computers executing the applications receives the visual form of the data from one of the plurality of different applications and tags corresponding to the visual form of the data. The visual form of the data as a vector image and the corresponding tags are stored into the database. An application on one of the plurality of computers accesses the tags and the visual form of the data from the database in response to queries on the database. Another application may be used to create documents including the visual form of data retrieved from the database, which may be stored into the database.

In another aspect, a vector image is processed to enable scaling of text in the vector image. Commands in the vector image corresponding to text are identified. Specified fonts in the identified commands may be changed to scalable fonts. Intercharacter spacing also may be added. Text rotations and scalings also may be modified.

In another aspect, compound documents may be created from the visual form of data from applications executed on a computer stored in a database. The visual form of information from the applications is captured into a database, by receiving the visual form of the data from the application, receiving tags corresponding to the visual form of the data, and storing the visual form of the data as a vector image and the tags into the database. The visual form of information from the applications is retrieved from the database. A compound document is created from the visual form of information retrieved from the database. The visual form of the compound document is captured into the database, by receiving the visual form of the compound document from the application, receiving tags corresponding to the visual form of the compound document, and storing the visual form of the compound document as a vector image and the tags into the database.

BRIEF DESCRIPTION OF THE DRAWINGS

In the drawings,

Fig. 1 is a block diagram illustrating an information storage and retrieval system for storing the visual form of data from an application into a database;

Fig. 2 is a block diagram illustrating operations of a print process in a Windows environment;

Fig. 3 is a schematic representation of a database table structure for the database;

Fig. 4 is a schematic representation of another database table structure for the database;

Fig. 5 is a schematic representation of another database table structure for the database;

Fig. 6 is a schematic representation of another database table structure for the database;

Fig. 7 is a block diagram illustrating the information storage and retrieval system of Fig. 1 with a service layer through which access to the database is coordinated;

Fig. 8 is a dataflow diagram illustrating sorting and filtering of the database;

Fig. 9 is a dataflow diagram illustrating copying records in a database;

Fig. 10 is a dataflow diagram illustrating deleting records from a database;

Fig. 11 is a dataflow diagram illustrating printing, previewing or sending reports from the query interface;

Fig. 12 is a dataflow diagram illustrating previewing of a report;

Fig. 13 is a dataflow diagram illustrating sending a report to another application;

Fig. 14 is a flowchart describing processing of a spool file;

Figs. 15A-B are a flow describing font processing of a spool file;

Fig. 16 is a flow chart describing text processing of a spool file; and

Fig. 17 is a flow chart describing how text data can be converted to character delimited data, in particular tab delimited data for insertion into a table for spreadsheets.

DETAILED DESCRIPTION

Many advantages may be obtained by receiving the visual form of data from a computer program and by storing it in a database. A visual form of data from an application or computer program is any image or portion thereof produced by the application or computer

program from the data for display or printing. An image is any data that define a matrix of picture elements (pixels) that may be displayed or printed on any display or printing device. Each pixel generally is defined by data from either a grey scale or a color palette. A vector image is an image specified by drawing commands. Example shapes include lines, polygons, objects, and text. The drawing commands are interpreted with respect to a target display or printing device to generate values for a matrix of pixels. In general, vector images may be transformed, e.g., scaled, rotated or flipped. A bit map image is an image specified by a matrix values specifying each pixel of the image. A bit map image also is known as a raster image. A bit map image may be included within a vector image.

In general, most applications transform data to a visual form as either a vector image or a bit map image for display on a display device. Applications also use function calls to the operating system to print a visual form of the data. In response to these function calls, the operating system typically produces a vector image that is translated by a printer into a bit map image to control the printer.

Referring now to Fig. 1, an information storage and retrieval system that captures the visual form of data from an application into a database will now be described. In Fig. 1, an application 50 is executed on a general purpose computer and is used by a user to create information which has a visual form that may be displayed or printed. This visual form is captured into a database 64 in the information storage and retrieval system in a format defined by the operating system 54 of the general purpose computer. The application 50 makes function calls 52 to the operating system 54 which cause the operating system to create data 56 representing the visual form of the data. Such data 56 is generally in the form of a vector image. Such function calls generally are provided by the operating system 54 in order to permit applications such as application 50 to print data to a printer. The data 56 in turn is read by a database storage module 58 and is stored as a report 60 with tags 62 in the database 64. The tags 62 include identifying information for the report, such as described below in more detail, which may include information provided by user input 57.

By relying on the output from functions calls to the Windows operating system made by an application in the process of performing a common operation, such as printing, the database storage module need not be designed to handle proprietary data formats from multiple applications. In general, for any operating system, the output of an operation from an application to another application in a standard format, such as printing, can be captured and

stored in a standard form in a database. Applications, such as application 50, need not be designed to handle communication with a database in order to store the visual form of data in a database.

After reports 60 are stored in the database, a database retrieval module 66 may access the database through queries 68 to retrieve one or more reports 70 and one or more tags 72. The database retrieval module 66 may be implemented to support any number of database management tasks, such as modifying, copying, moving, deleting, reading and otherwise manipulating the database records, as described in more detail below. A query interface 76 permits a user to input parameters for searching, sorting and filtering the reports in the database. Collections of tags associated with filtered or sorted reports are provided to the query interface 76 for review by the user. A selected report may be provided to a report previewer 80 or another application 84 through a command available through the query interface 76.

A report previewer 80 receives reports 78, and tags 74, through the database retrieval module 66 to display a report to the user. The report previewer, as described in more detail below, permits a user to provide input 81 to select a report or a portion of a report 82 to be transferred or sent to another application 84. Using this other application 84, a compound document may be compiled from many reports stored in and retrieved from the database. This compound document also may be stored in the database 64 in the same manner that application 50 stores data into the database.

The various components of the information storage and retrieval system such as shown in Fig. 1 may be implemented as a computer program using a general purpose computer system. Such a computer system typically includes a main unit connected to both an output device which displays information to a user and an input device which receives input from a user. The main unit generally includes a processor connected to a memory system via an interconnection mechanism. The input device and output device also are connected to the processor and memory system via the interconnection mechanism.

It should be understood that one or more output devices may be connected to the computer system. Example output devices include a cathode ray tube (CRT) display, liquid crystal displays (LCD) and other video output devices, printers, communication devices such as a modem, and storage devices such as disk or tape. It should also be understood that one or more input devices may be connected to the computer system. Example input devices include

a keyboard, keypad, track ball, mouse, pen and tablet, communication device, and data input devices.

The computer system may be a general purpose computer system which is programmable using a computer programming language, such as "C++," Visual Basic, JAVA or other language, such as a scripting language or even assembly language. In a general purpose computer system, the processor is typically a commercially available processor, such as the series x86, Pentium and Cerelon processors, available from Intel, similar devices from AMD and Cyrix, the 680X0 series microprocessors available from Motorola, and the PowerPC microprocessor from IBM. Many other processors are available. Such a microprocessor executes a program called an operating system, of which WindowsNT, Windows95 or 98, UNIX, Linux, Solaris, OS/2,.DOS, VMS, MacOS and OS8 are examples, which controls the execution of other computer programs and provides scheduling, debugging, input/output control, accounting, compilation, storage assignment, data management and memory management, and communication control and related services. The processor and operating system define a computer platform for which application programs in high-level programming languages are written.

A memory system typically includes a computer readable and writeable nonvolatile recording medium, of which a magnetic disk, a flash memory and tape are examples. The disk may be removable, known as a floppy disk, or permanent, known as a hard drive. A disk has a number of tracks in which signals are stored, typically in binary form, i.e., a form interpreted as a sequence of one and zeros. Such signals may define an application program to be executed by the microprocessor, or information stored on the disk to be processed by the application program. Typically, in operation, the processor causes data to be read from the nonvolatile recording medium into an integrated circuit memory element, which is typically a volatile, random access memory such as a dynamic random access memory (DRAM) or static memory (SRAM). The integrated circuit memory element allows for faster access to the information by the processor than does the disk. The processor generally manipulates the data within the integrated circuit memory and then copies the data to the disk when processing is completed. A variety of mechanisms are known for managing data movement between the disk and the integrated circuit memory element, and the invention is not limited thereto. It should also be understood that the invention is not limited to a particular memory system.

Such a system may be implemented in software or hardware or firmware, or a combination of the three. The various elements of the system, either individually or in combination may be implemented as a computer program product tangibly embodied in a machine-readable storage device for execution by a computer processor. Various steps of the process may be performed by a computer processor executing a program tangibly embodied on a computer-readable medium to perform functions by operating on input and generating output. Computer programming languages suitable for implementing such a system include procedural programming languages, object-oriented programming languages, and combinations of the two.

It should be understood that the invention is not limited to a particular computer platform, particular processor, or particular programming language. Additionally, the computer system may be a multiprocessor computer system or may include multiple computers connected over a computer network. It should be understood that the components shown in Fig. 1 and their subcomponents illustrated in the remaining figures may be separate modules of a computer program, or may be separate computer programs, which may be operable on separate computers. The data produced by these components may be stored in a memory system or transmitted between computer systems.

The system shown in Fig. 1 may be implemented using a computer network, such that the application 50, operating system 54 and database storage module 58 are executed on one computer, the database 64 is executed on another computer and the database retrieval module 66, report previewer 80 and the other application 84 are executed on yet another computer. There may be many computers with a database retrieval module 66, many computers with a database storage module 58 and many databases 64. One computer may have both a database retrieval module and a database storage module for use with its applications 50 and 84. One computer also may have the database retrieval module, the database storage module and the database along with the applications 50 and 84. The various possible configurations of computers in a network permit many users to create compound documents from multiple sources of data, even if the data is created, used and shared over a period of time by multiple different users with multiple different computer programs that are dispersed geographically.

The various modules of Fig. 1 will now be described in more detail. The applications 50 and 84 may be any applications that may be executed by the operating system 54 of the general purpose computer. Example kinds of applications include, but are not limited to,

laboratory equipment control and data analysis programs, word processing programs, graphics programs and spreadsheet programs.

The database 64 may be any kind of database, including a relational database, object-oriented database, unstructured database or other database. Example relational databases include Oracle 8i from Oracle Corporation of Redwood City, California, Informix Dynamic Server from Informix Software, Inc. of Menlo Park, California, DB2 from International Business Machines of Yorktown Heights, New York, and Access from Microsoft Corporation of Redmond, Washington. An example object-oriented database is ObjectStore from Object Design of Burlington, Massachusetts. An example unstructured database is Notes from the Lotus Corporation, of Cambridge, Massachusetts. A database also may be constructed using a flat file system, for example by using files with character-delimited fields, such as in early versions of dBASE, now known as Visual dBASE from Inprise Corp. of Scotts Valley, California, formerly Borland International Corp.

The operating system 54 may be, for example, Windows95, Windows98 or WindowsNT from Microsoft Corporation. Using this operating system, the database 64 may be accessed through the Open Database Connection (ODBC) protocol. Other example operating systems are noted above. The invention is not limited to any particular operating system nor to those mentioned herein. The operating system used in any particular embodiment affects which commands an application may use to cause the operating system to generate a visual form of data for access by either a print driver or other application. In the following description, the Windows95, Windows98 and WindowsNT operating systems are only illustrative. Other operating systems may be used with corresponding changes to how their output is used and how control is coordinated among the modules of the system according to the specification of the operating system. The following description illustrates how function calls from an application to the Windows operating systems, e.g., print commands, may be used to transfer data from the application into a database. Other function calls to the operating system also may be used. Other mechanisms through which the visual form of data is transferred from one application to another in a standard form also may be used.

Using the Windows95, Windows98 and WindowsNT operating systems, the data 56 representing the visual form of the data output by the operating system in response to function calls from an application to print the data is in a Windows Metafile format, according to

Microsoft. A metafile is a vector image, or, a list of commands, draw objects, text, and commands to control style. Theoretically, a metafile may be used in any Windows application. A Windows metafile (WMF) is a 16-bit metafile that is supported by Windows 3.1. An enhanced metafile (EMF) is a 32-bit enhanced metafile that is supported by Windows 95, Windows 98, and Windows NT having a super set of WMF commands.

Referring now to Fig. 2, the operation of printing in the Windows operating systems and its use to capture the visual form of data from an application into a database will now be described. In order to print on a printer in a Windows environment, the printer has an associated print driver 90. When the printer is installed, the operating system is informed of the location of the print driver, i.e., its file name. The print driver specifies the characteristics of the printer to the operating system.

An application 50 (as in Fig. 1) permits a user to select a printer through a user interface, such as a graphical user interface with menus. The selected printer also may have various printing options that may be selected. Through a function call made by the application in response to user input, the user may invoke a user interface 94 for the print driver to permit the user to specify user information and printing preferences. Given a selected printer, preferences and information to be printed, the application 50 issues function calls to a Graphics Device Interface (GDI-32) 92, which is part of the Windows operating system. The GDI-32 requests the selected print driver 90 and its user interface for information about the printer that in turn is given back to the application 50, and is retained by the GDI-32, to assist in the process of generating a correct sequence of function calls to the operating system to print the selected information.

The GDI-32 outputs data into spool files 98 and makes function calls with the names of the spool files to the spooler process 96 to queue the spool files for printing on their designated printers. A spool file in the Windows operating system is designated as a Windows metafile by Microsoft. A printer spool file is not a true metafile, however, because it actually contains printer setup data in addition to any referenced or embedded metafiles. In Windows95, a spool file contains the file names of any metafiles. In WindowsNT, the metafiles are embedded in the spool file. In both cases, there is one spool file per printed document, and each page of a document has a separate metafile.

The spooler process 96 is informed about the location of a print processor 100 associated with the selected print driver 90. The spooler process 96 calls the print processor

to process any spool files that the spooler process has queued for the print processor. Generally, a typical print processor receives the spool file 98 from the spooler process 96 and converts it to a format used by the printer, such a printer control language (PCL), PostScript or other, typically proprietary, format. Instead of printing, the print processor 100 in Fig. 2 causes the vector image data produced by the operating system to be formatted, associated with tags, and stored in a database.

More details about metafiles, print drivers, print processors, spooler processes and spool files are available through the Microsoft Developer Network and the Microsoft Development Network Library accessible through the Internet.

In one implementation shown in Fig. 2, the print processor 100 spawns two processes. The first process, tag data acquisition process 104, described in more detail below in connection with Fig. 7, obtains the tag information from the user for the report being stored in the database. This process provides a user interface to the user to allow the user to input data for the tags and stores the tags in a file 105 having a predetermined name. The user may enter tags for each report or may indicate a request for batch printing in which case the tag data acquisition process uses an initial set of tags to compute tags for each report to be stored in the database without user interaction. The second process, database access process 106, described in more detail below also in connection with Fig. 7, obtains both the tag information from the tag data acquisition process 104 through the file 105 and the names of the metafiles 108 to be stored. The tags and the metafiles are processed and sent to the database 64.

Although the print processor 100 may read the data from the spool file and store that data in the database with the tags, it has been found that in some instances a vector image, e.g., a Windows metafile, in the spool file might not be scalable. In other ways noted above, the spool file also is not a true Windows metafile. Therefore, a metafile processor 102, described in more detail below in connection with Figs. 14 through 16, responds to function calls from the print processor 100 to process the spool file into metafiles 108, to improve their scalability and otherwise reformat the spool file into one or more Windows metafiles, one for each page of the report or document being printed. The file names of the metafiles 108 resulting from processing the spool files are made available to the database access process 106, either through the print processor 100 or directly by the metafile processor 102.

The print driver 90, print driver user interface 94, print processor 100, metafile processor 102, tag data acquisition process 104 and database access process 106 form the database storage module 58 in Fig. 1.

Having now described how the print driver and print processor store the visual form of data from an application in a database, the database in which this data is stored will now be described in more detail. The database stores the reports and tags using a structure appropriate to the selected database type. The kinds of information that may be stored, i.e., the tags, may include any combination of, for example, information about the report, computer environment information, user information, organizational information and date and time information. Various other information also may be stored as desired with appropriate changes to the database structure. Example information about the report includes a name for the report, a number of pages and a project for which the report is prepared, user comments, the source application and any batch identifier. Example computer environment information includes the name and/or version of the application that created the report, a machine identifier, machine location, operating system and any group or domain name for the machine. Example user information includes a user identifier, such as a login name, and contact information. Example organizational information includes the user's title and the name of a group to which the user belongs.

Figs. 3-6 illustrate an example database schema for storing the reports and tags in a relational database. It should be understood that any other database schema may be used and that the following example is merely illustrative. The variable types are specified for Oracle and Microsoft Access databases. It should be understood that other databases could be used and that these variable types would be defined according to the specifications and operation of the other databases.

Fig. 3 illustrates a main table 300 that contains records with the tag information for each report. In this table, each report is printed into the database and added as a record, with each record being assigned a unique identifier, LTId. The LTId may be generated by incrementing and updating the value stored in field 602 in Fig. 6, LTLastId, described below. Main table 300 includes in field 1, 302, LTId, an integer type variable in both Access and Oracle, which is the report identification number. In field 2, 304, the LTVerson is an integer type variable in both Access and Oracle, and is the version of the database schema in use when the record was inserted into the database. In field 3, 306, LTPrjDesc is a Varchar(50)

type variable in both Access and Oracle, and is a 50 character description of the project. In field 4, 308, LTRepNme is a Varchar(30) type variable in both Access and Oracle, and is a 30 character description of the report name. In field 5, 310, LTRepId is a Varchar(150) type variable in both Access and Oracle, and is a 150 character description of the report identification name. In field 6, 312, LTBtchId is a Varchar(30) type variable in both Access and Oracle, and is a 30 character description of the batch identification. In field 7, 314, LTUsrNme is a Varchar(30) type variable in both Access and Oracle, and is a 30 character description of the user name. In field 8, 316, LTUsrCmnt is a Varchar(254) type variable in both Access and Oracle, and is a 254 character description of any user comments. A user may insert keywords or other information in this field to facilitate later retrieval. In field 9, 318, LTPrtDat is a Varchar(30) type variable in both Access and Oracle, and is up to 30 characters identifying the print date. In field 10, 320, LTPrtTim is a Varchar(30) type variable in both Access and Oracle and is up to 30 characters identifying the print time. In field 11, 322, LTPrtApp is a Varchar(50) type variable in both Access and Oracle, and is a 50 character description of the of the name of the source application. In field 12, 324, LTMachNme is a Varchar(50) type variable in both Access and Oracle, and is a 50 character description of the machine name on which the source application was executed. In field 13, 326, LTDomnNme is a Varchar(50) type variable in both Access and Oracle, and is a 50 character description of the domain or workgroup name in which the machine is located. In field 14, 328, LTLogNme is a Varchar(50) type variable in both Access and Oracle, and is a 50 character description of the login name of the user of the source application. In field 15, 330, LTXExtnt is an integer variable type in both Access and Oracle and is X extent of pages in the report. In field 16, 332, LTYExtnt is an integer variable in both Access and Oracle and is the Y extent of pages in the report. In field 17, 334, LTNumOfPg is an integer type variable in both Access and Oracle and is the number of pages in the report. In one embodiment, if this field is left blank then the number of pages is computed for the LTId from the table in Fig. 4. In field 18, 336, LRCRC32 is an integer type variable in both Access and Oracle and is the CRC of the tags and numeric data in the record. In one embodiment, if the calculated value of the CRC does not match the stored value, this is indicative that the data is corrupted.

Fig. 4 illustrates a page table 400. Page table 400 includes records that help identify the number of pages in a report. In one embodiment, each page is identified using a page number, with a sequence of page numbers beginning with one. In one embodiment, the table

is indexed by combining the LTId, record 1, 302, and the page number. In field 1, 302, LTId is an integer type variable in both Access and Oracle, and is the report identification number. In field 2, 402, LTPgNum is an integer variable type in both Access and Oracle that represents the page number of a record. In one embodiment, the LTId and LTPgNum constitute a unique record identifier. In field 3, 404, LTPgInfo is a Varchar(100) type variable in both Access and Oracle and is a 50 character description of the page information. This record may be omitted.

Fig. 5 illustrates a metafile data table 500. The metafile data table 500 includes the records that hold the metafiles. In one embodiment, each printed page is a metafile that is stored in this table. In one embodiment, the metafile can be compressed and encoded and this is indicated by a particular record identifying the buffer type. Data table 500 can include in field 1, 302, LTId, an integer type variable in both Access and Oracle, which is the report identification number. In field 2, 402, LTPgNum is an integer variable type in both Access and Oracle that represents the page number of a record. In field 3, 502, LTMFRNum is an integer variable type in both Access and Oracle and represents a metafile record number. In one embodiment, the LTId, LTPgNum, and LTMFRNum constitute a unique record identifier. In field 4, 504, LTMFBfTY is an integer variable type in both Access and Oracle that represents a metafile buffer type. In one embodiment, the buffer type is used to particularly identify the type of metafile record stored and in one embodiment can be a regular, enhanced, base-64 encoded, or other form of metafile record. In field 5, 506, LTMFBfLn is an integer variable type in both Access and Oracle that represents the length of data in the LTMFBufr, field 6, 508. In field 6, 508, LTMFBufr is a memo variable type in Access and a long variable type in Oracle that includes a piece of the metafile record stored in blocks. Each block has a size, such as 16K bytes.

Fig. 6 illustrates a control table 600. Control table 600 includes in field 1, 602, LTLastId, which is an integer variable in both Access and Oracle that represents the last LTId inserted in the database. The value may be kept the same if a record is deleted instead of decremented. The value may be reset to 0 if all the records are deleted to allow the new entries to start from 1. The value may have a maximum value, such as 4, 294,967, 295.

Having now described the database and how it is structured, access to the database for storing and retrieving information will now be described. For several reasons, a program called a "service layer" is provided through which other applications access the database. For example, the service layer isolates applications from the complexities of the database by

providing simple function calls that other applications may make to the service layer to perform various database operations. The service layer performs operations on the database, for example through ODBC calls, in response to a function call from an application. Also, by providing error checking and data encoding in the service layer, any altered data in the database can be detected.

Fig. 7 illustrates in more detail how the print driver 90, print processor 100, metafile processor 102 and print driver user interface 94 of Fig. 2, shown as the "print driver" 204 in Fig. 7, interact with the service layer 200. The operation of the service layer 200 when retrieving information from the database will be described in more detail in connection with Figs. 8 through 13. In Fig. 7, the service layer 200 receives a report (metafiles 108) created by the print driver 204 and the tag data through database access process 106 from files 105 which specify the tags and the names of the files 108 containing the metafile data for the reports. This data is stored by the service layer through calls to the ODBC layer 206 into the database 64. The interaction of the print driver 204, database access process 106 and tag data acquisition 104 with the service layer 200 will now be described in more detail.

In general, as noted above, the print driver 204 causes metafiles 108 to be created from a spool file 98, with the metafile names stored in files 105. The tag data also are stored in files 105 by tag data acquisition process 104. The tag data acquisition process is spawned by the print driver 204 to obtain the tag data from the user for this report. Tag data acquisition module has two modes of operation: batch and manual. This mode of operation may be set by a user for a given application, or for all applications, through the print driver user interface 94. This setup information is stored in files 105. When the tag data acquisition process is spawned, the print state is retrieved from the setup parameters. If the print state for this application is batch printing, a set of tags is computed from any previously stored tags for this application. Otherwise, a dialog box is displayed to the user to request the user to input tag information. After the user has input the tag information or after the tag information is computed, the tag information for this application is stored in files 105.

The database access module 106 is spawned by the print driver 204 to store tags and metafiles into the database. In particular, database access module 106 opens the database and handles any authentication that may be required by the database. To perform this operation, the database access process 106 issues a request to the service layer 200 which, in turn, issues a request through the ODBC layer 206 to the database to open and authenticate this user for

the database and begin the database transaction. After successful opening and authentication, database access process 106 issues a request to the service layer 200 to store the metafiles specified by the metafile names from file 105. In response to this request, the service layer retrieves the metafiles 108 corresponding to the metafile names received from the database access process 106 and stores them in the metafile table in the database by issuing requests through the ODBC layer 206. The database access process 106 requests the service layer to store the tag information by providing the tags to the service layer 200. The service layer 200 writes the tag information to the main table in the database by issuing requests through the ODBC layer 206. The database access process 106 closes the database by issuing a request to the service layer 200. The service layer 200 then commits the transaction in the database and closes the database through requests to the ODBC layer 206.

The service layer 200 also may provide both more efficient storage and data integrity checking while storing a report and tags in the database. For example, as shown in Fig. 7, the incoming visual form of the data may be compressed using a data compression algorithm, of which several are known in the art. A lossless compression algorithm is preferable, such as available in the Crusher software library from DC Micro Development. An error checking calculation, such as a cyclical redundancy check (CRC) operation, may be used to compute a CRC value for the compressed data. When reading data from the database, the CRC value for the read data may be calculated and compared to the stored value to detect whether the data has been corrupted. The tag data also may be processed to compute a CRC value for the tag data. When reading the tag data from the database, the CRC value is checked to verify the integrity of the tag data. The compressed visual form of the data also may be encoded to allow storage in databases that do not support storage of 8-bit binary data. An example encoding technique is base 64 encoding. The encoded data is split into blocks of 16 kilobytes. A database record is written for each block of each page of the encoded data.

The service layer also acts as an intermediary between applications and the database to permit an application to search the database in response to user queries and to retrieve data from the database. The operations of the service layer in connection with two other applications that use the service layer will now be described in more detail in connection with Figs. 8 through 13.

In general, the service layer 200 reverses operations performed on the data when the data was written to the database. For example, the service layer decodes the data, checks the

CRC value to ascertain data integrity and decompresses the data prior to passing the data to another application. Similarly, the tag data is retrieved and the CRC value is checked before the tag data is passed to another application. The service layer also queries the database according to queries provided from another application.

Fig. 8 illustrates the interaction between the query interface 76 (Fig. 1), service layer 200 (Fig. 7) and the database when a user selects a database to view and specifies a sort or filter operation to be applied to the database records. The query interface 76 receives a request from a user to open a database. In response, query interface 76 passes the request to the service layer. The service layer issues a request to the ODBC layer to obtain a list of databases, each having a name. The service layer receives this list and provides it to the display for presentation to and selection by the user.

After a user selects a database, and optionally specifies a filter or sort operation, the query interface issues a request to the service layer to open the database. Any filter or sort operation may be defined according to the database structure and the query language used to access the database. The service layer issues a request through the ODBC layer 206 to the database to open the database for this user. The database may prompt the user for a username and password. The service layer prepares a query based on any sort or filter parameters specified by the user. This query is issued to the database and applied to the main table. A set of records is received from the database by the service layer and the CRC values of the tags are verified. The set of records is placed in a list, with each entry additionally having a flag to indicate whether the CRC values of the tags for the record were correct. The list is returned to the query interface for presentation to the user. When displayed, the query interface may provide a visual indication to the user of whether any record has been altered based on the CRC flag. The list may have a limited number of entries, in which case the service layer stops reading records from the database and saves a cursor when the list is full.

The user may navigate through the list, for example, by scrolling through the list or by scrolling through pages of the list. The query interface, in response to scrolling commands from the user, requests the service layer to access more records from the database as needed. The user also may decide to refresh the list. The query interface, in response to a refresh command, issues a refresh command to the service layer. The service layer in turn invalidates any saved cursor and creates a new list of records in the same manner as described above.

The user also may decide to change the sort or filter parameters. The query interface, in response to a change in the sort or filter parameters, sends the new parameters to the service layer, which stores the parameters. The query interface then issues a refresh command to the service layer and receives a new set of tags.

The user may terminate usage of the query interface by closing the database, which causes a request to be sent by the query interface to the service layer. The service layer in turn invalidates any stored cursor and issues a close command to the database through the ODBC layer.

Other database management functions can be performed by the user in addition to searching the database using sort and filter commands. Such functions include copying records from one database to another, deleting records from a database, moving records from one database to another (performed by copying the records and deleting them), and merging databases together (performed by copying all the records from two or more databases into another).

Copying of records from one database to another will now be described in connection with Fig. 9. It is assumed that a user has already opened the database and is viewing a set of reports. The query interface 76 receives the user selection of one or more records, and a command from the user to copy the selected records to a database. The selected records are indicated to the service layer by the query interface. The display then prompts the user for a database name to be opened, in a manner described above. The database to which the records are to be copied is opened through the service layer and the database may prompt the user for authentication.

The query interface issues a request to the service layer to copy the selected records. In response, the service layer invalidates any saved cursor and prepares query statements based on the selected reports and pages as a filter. The query is executed on the metafile, page and main tables of the database. For each record in the main table, a new report identifier is computed for the destination database using its control table. The records retrieved from the database are copied to the corresponding main, page and metafile page tables of the destination database. The metafiles are not decompressed or decoded to save processing time. A new CRC value is computed for the tag fields to account for the new identifier for the report before sending the new tags to the destination database. The report list from the current database is refreshed in the query interface in the manner described above.

Deleting records from a database will now be described in connection with Fig. 10. It is assumed that a user has already opened the database and is viewing a set of reports. The query interface 76 receives the user selection of one or more records, and a command from the user to delete the selected records from the currently open database. The selected records are indicated to the service layer by the query interface. The authentication used by the user when the database was opened is verified with the user. The database may prompt the user for new authentication.

The query interface issues a request to the service layer to delete the selected records. In response, the service layer invalidates any saved cursor and prepares delete statements to be issued to the database based on the selected reports and pages as a filter. The query is executed on the metafile and page tables of the database to delete the records. A delete statement is prepared for the main table of the database using the selected reports as a filter. This statement is executed on the main table to delete the records for the report. The report list from the current database is then refreshed in the query interface in the manner described above.

After a user has already opened the database and is viewing a set of reports, the user may select one or more reports to be printed, previewed or sent to another application. Such printing, previewing and sending of reports from the database will now be described in connection with Figs. 11-13. The query interface 76 receives the user selection of one or more records, and a command from the user to print, preview or send the reports associated with the selected records. The selected records are indicated to the service layer by the query interface. The metafiles for the selected records then are requested from the service layer by the query interface.

The service layer invalidates any saved cursor and prepares a query statement with the selected reports and pages as a filter. The query is executed on the metafile table of the database. When all of the records for a page are read, the pieces of the records are reassembled, decoded and uncompressed, and the CRC value for the page is verified. The metafile for each page of each report is stored in a named temporary file as indicated at 78. A list of report identifiers, page numbers and metafile names is constructed and returned to the query interface. The query interface processes the list according to whether the command was a print, preview or send, in a manner that will now be described.

If the command was a print, the display simply plays the temporary metafile(s) to a device context for the specified printer using the operating system.

If the command was a preview, the query interface operates in a manner to be described in more detail in connection with Fig. 12. In particular, a control file 1200 with a predetermined name is created including the tags and names of the temporary files containing the metafile for each page of each report to be previewed. The query interface spawns the report previewer 80, a separate program, which in turn accesses the control file to read and display the reports. The report previewer 80 displays the reports by playing the metafiles to a device context for the display, i.e., by issuing a play metafile command to the operating system.

The previewer may perform many kinds of operations on the played metafile data, including permitting the user to scroll through the display and view different pages of the report. The previewer also may permit a user to find instances of specified text in the report. However, searching for text in a particular order in a metafile is complicated by the fact that the TEXTOUT commands in the metafile may not occur in the same order in the metafile as the corresponding text would appear when displayed. Accordingly, the report previewer processes the text in each selected metafile into an ordered list so that words may be searched in the order that they appear within each metafile. Such ordering also permits any previous or next occurrences of a word or other character string to be identified in the metafile and among a set of metafiles. Documents from many different applications may be searched together using this technique.

Ordering text in metafile is performed in the following manner. Each TEXTOUT command is identified in the metafile and its corresponding coordinates are obtained. The variety of TEXTOUT commands are then sorted by page (i.e., metafile), then by Y position in the display area, then by X position in the display The sorted commands are converted into a doubly linked list of records of text strings and the position of the text string in the display. Any text strings that are close together, within a percentage of the text height, may be concatenated in this structure. Any efficient sorting algorithm may be used to perform this sort information. For example, an insertion sort may be used.

Both the query interface and the report previewer may permit a user to send a metafile 78 (or in the case of the previewer, a selected portion of a metafile 82) to another application 84. The send operation also may send a report including all graphics, including text only, or

including text transformed into a table (which is described in more detail below in connection with Fig. 17). This operation will now be described in connection with Fig. 13. This operation allows data to be selected and inserted into documents being authored using other applications.

From the query interface, the process of sending a report is similar to previewing. In addition, the user is prompted for the name of the destination application to which the metafile will be sent. A control file 1300 is created with metafile names, the name of the destination application and a set of keystrokes defining how a paste is performed in the destination application. A separate computer program 1302 called "Send" is executed instead of the report previewer, which operates on the control file 1300 in a manner described below to paste the report in a currently opened document in the destination application 84.

The "Send" program also may be executed by the report previewer 80. The user provides the name of the destination application. The report previewer creates a new temporary metafile 82 based on the user's selection, in a manner described below. A control file 1304 similar to the control file 1300 created by the query interface is created and the "Send" program 1302 is executed in the same manner.

The appropriate keystrokes for the Send program to issue to the destination application may be determined in many ways. For example, the menus of the application may be searched for the command "Paste" to identify the corresponding keystrokes. Alternatively, the keystrokes for various applications may be determined in advance and saved in a file accessible by the Send program. As a default, because the keystroke for the paste function in a user interface for a standard Windows application is Ctrl-V, this keystroke also could be used.

How the report previewer creates a new metafile based on a user's selection of a portion of a metafile will now be described. A user's selection generally is represented by coordinates defining a rectangle in the area in which the metafile is displayed. Each command of the metafile is then compared to the defined coordinates. Any commands that do not have associated coordinates are automatically placed into the new metafile. Any commands that produce display information outside of the specified coordinates are discarded. Any commands that produce display information inside the specified coordinates are retained. In some cases, a command may produce display information both inside and outside the specified coordinates. These commands, such as line segments and bit maps, may be readily

modified to be constrained within the specified coordinates using known techniques. The resulting set of metafile commands are stored in named temporary files whose filenames are listed in the control file.

The "Send" program 1302 will now be described. This program first reads the control file 1300 or 1304 using its predetermined name. The "Send" program 1302 does not need to know whether it was invoked by the query interface 76 or the report previewer 80. Any 32-bit metafiles (EMF) specified in the control file may be converted to 16-bit metafiles (WMF) 1310 through a request to a metafile processor 1306. The conversion also may be performed automatically by the Windows GDI-32 if only the EMF files are placed on the clipboard. The conversion performed by the metafile processor 1306 deletes "color space" records, any intercharacter spacing in "TEXTOUT" records (described below) and any "modify world transform" records that contain illegal parameter values.

Clipping records created by Windows95 also may be modified. In particular, the viewport and window origins are tracked to maintain current values. The boundaries of clipping regions are adjusted, using known techniques, to account for the current viewport and window origins. Some clipping records also cause text to be incorrectly hidden when the metafile is played. Any clipping record that affects a text record is modified or removed if necessary to prevent text from being incorrectly hidden.

The 16-bit and 32-bit metafiles both are placed on the "Clipboard" 1308 through clipboard placement 1312 which makes a function call to the operating system. The clipboard is an area of memory in which data may be stored temporarily and from which data may be read by any application. In order to place data on the clipboard, a command such as "cut" or "copy" is used. A "paste" function generally is provided by most applications to read data from the clipboard to place the data at the current location in the document currently being processed by the application.

The Send program then sends the keystrokes to the destination application via the operating system to cause the application to paste the data from the keyboard into its current document.

The send operation described in connection with Fig. 13 is not limited to the implementation shown and can be provided in many ways. For example, the query interface and report previewer are described above as determining the set of keystrokes for a given application. These keystrokes also may be determined or may be accessed by the Send

program directly and not through control files 1300 or 1304. Alternatively, the entirety of the Send program 1302 may be implemented in the query interface 76 and the report previewer 80. The send operation also may be provided to send metafiles from one source application to another destination application if the source application enables third parties to develop "plug-in" macros or libraries to add new functions to the source application's menus or toolbars. An example source application that has such capability is Word from Microsoft Corporation. In such an application, the combined functionality of the previewer or query interface and the send program would be invoked by the source application to transmit metafile data directly to the destination application.

As noted above, the spool files 98 are not true metafiles. The visual form of the data provided by the operating system in response to an application, for example through printing, also may become distorted if transformed, for example, if scaled or rotated. Such distortions generally are attributable to the fact that the output of the operating system, for example in response to a print command, generally is not intended to be transformed. An application typically is implemented so that the output of the printer in response to the operating system output matches the visual form of the data. Generally, applications are not tested for whether the commands sent to a printer can be scaled without distortion. Therefore, the visual form of the data 56 (See Fig. 1) output by the operating system 54 may need to be converted into a correct form of metafile that can be transformed without producing distortions.

Such a conversion will now be described in connection with Figs.14 through 16, again using the Windows operating system and the Windows Metafile as an example. Referring now to Fig. 14, spool file 98 is received by the metafile processor 102 in step 800. The metafile processor, in step 802, opens a new file for each page in the spool file in which the new metafile(s) will be stored. A private record is inserted into this new file in step 804 to provide version information. For example, the operating system from which the application was printed and the version of the metafile processor program may be stored. This information permits a metafile to be processed differently according to the version information.

The "DEVMODE" structure available in Windows is copied and stored into a private record in the metafile in step 806. This data structure describes all of the characteristics of the printer and the print job, including page size, resolution and orientation. Any similar information available from other operating systems also may be stored. This record may be

omitted if the information is provided accurately and completely in a header of the metafile(s) received from the spooler process.

In step 808, the metafile for a page obtained from the spool file is converted into a revised metafile for a page to improve its scalability according to the specifications for these data formats provided by the Microsoft Corporation, in a manner described in more detail below. During this step, the metafile processor inserts into the output file a comment record containing any original command, followed by the modified or new command. Retaining the original command facilitates in reconstructing the original for conversion to another format.

In step 810, the metafile processor inserts an end-of-file comment record at the end of the enhanced metafile data. The end-of-file comment record is used to delineate any boundary between metafiles contained within another metafile. Such metafiles can be thought of as "nested metafiles." When processed, a metafile can have different properties and characteristics depending upon the operating system platform and application with which it was created. For example, metafiles created on Windows NT can have different characteristics that those created on Windows 95. In order to ensure proper processing, the start and end points of each nested file are provided by the records inserted in steps 804 and 810.

In step 812, the metafile processor closes the new metafile file and provides the file which stores the filenames in files 105.

The conversion of commands in a metafile performed in step 808 will now be described in more detail. Various modifications are made because of differences between metafile types for different operating systems, and because metafiles created by printing by different applications generally are not designed to be scalable, and therefore sometimes are not.

Commands that produce potential resource leaks when a metafile is played, such as create font or other create commands, are identified. These commands may be processed to ensure that the resource is properly freed up.

Some applications generate metafiles that attempt to display text in coordinate transformation spaces that are incompatible with proper display of scaled, rotated or otherwise transformed text, such as a negative Y axis direction and incompatible mapping modes. The result is text that is incorrectly oriented or incorrectly scaled in the vertical and/or horizontal directions. The metafile processor detects this condition by identifying commands with the

negative Y axis or an incompatible mapping mode. Before the record that displays the text, a metafile record is added to modify the coordinate transformation space temporarily. The new coordinate transformation is determined from the specified transformation, the coordinates of the text to be placed, the font orientation specified, the aspect ratios and the mapping mode, using known techniques. The coordinate transformation space is modified to preserve the intended position, orientation and aspect ratios. Another metafile record is added after the record that displays the text, to reset the coordinate transformation space to its original state.

Some metafiles generated by applications running on Windows NT contain records to "set miter limits." The metafile processor removes these records because they are not recognized by the Windows 95 GDI-32. The effect on the resulting image is negligible.

Some metafiles contain 16-, 24-, or 32-bit color bitmaps that contain color palettes. The metafile processor modifies the bitmap records by removing the color palettes because they are not needed and because they cause distortion of the bitmap when displayed.

Some metafiles display text aligned relative to the current cursor position causing the text position to be incorrectly determined in the viewer program and causing the text to disappear when printed to some devices. The metafile processor changes the text alignment record to specify alignment based on absolute coordinates rather than relative coordinates and then modifies each metafile record that displays text to contain the correct absolute coordinates, which may be determined using known techniques from the current cursor position, which can be obtained through a function call to Windows.

Font and text commands in general also are processed to improve their scalability in a manner described below in connection with Figs. 15A-B and 16.

Font processing is described in Figs. 15A and 15B. Some applications generate metafiles that use non-scalable fonts. These metafiles are look correct when printed to a sheet of paper but are distorted when displayed at scalings different from that of a printer or on devices other than a printer. The metafile processor adds metafile records to specify scalable fonts to prevent this distortion. In steps 1602-1608 a series of decision points determines whether or not a TrueType font has been used in the metafile. In step 1602 if no font has been specified in the metafile, then a default TrueType font is selected, step 1610. In step 1604 if a System font has been selected, then a TrueType font is selected to replace it, in step 1610. In step 1606 if a bitmap font has been selected, then a TrueType font is selected to replace the bitmap font, in step 1610. In step 1608 if the application selects a font (sometimes called

stock fonts) through the Windows SelectObject function, then a TrueType font is selected to replace the font selected through SelectObject, in step 1610. After completion of steps 1602-1610, two flags in the metafile are set to true to indicate that only proof quality TrueType fonts should be used in step 1612. In step 1614, if a font size has not been specified, then a default font size is selected, step 1616. The user previously may have selected font size and face names other than default values. Those values may be used in place of the default values. In step 1618, if an axis rotation or font rotation is encountered, which can cause a distorted output graphic, new values of the rotation or rotations are calculated in step 1620, in the manner described above.

Text processing will now be described in connection with Fig. 16. Some applications generate metafiles that contain text records that do not specify intercharacter spacing. The metafile processor calculates the proper intercharacter spacing values and modifies the text records using this process. Each metafile command is evaluated to determine if it is a TEXTOUT record. A TEXTOUT record is specified as containing an array of character spacing values. That is, for every character in a string to be printed, a value is supplied that indicates an amount of space to leave between it and the next character. This distance value might be absent. Therefore, when a TEXTOUT record without distance values is encountered, the correct distance values are obtained in step 1704, for example by a function call to the GDI-32. In step 1712 the old TEXTOUT record is replaced with the newly calculated data. The new record may include an additional field to indicate that it is an added record.

Additional text processing also may be performed in order to index any text in the metafile and store such an index as additional tag information for a report in the database. This index also may be represented as a doubly-linked list that could then be used by the report previewer 80 described above to permit text searching within the previewer.

Another form of text processing that may be performed, but in this case by the report previewer 80, is a conversion of text data, such as produced by TEXTOUT commands from a metafile, into character delimited data, such as tab-delimited data, which then may be converted into a table or spreadsheet. This process, described in connection with Fig. 17, may be used for any form of text data, including but not limited to, text data from a metafile within TEXTOUT commands. The process involves receiving selection coordinates specified by the user, for example, created by the user when selecting text displayed on the display. The

selection coordinates generally define a rectangle within the display area. After the selection coordinates are received in step 1800, the text content within the selected coordinates is identified in step 1802. This step may be performed by playing each metafile command and comparing the coordinates of each command to the selection rectangle, or by examining the coordinates within a data structure such as the doubly-linked list described below.

The columns defined by the selected text are then detected. These columns are detected first by initializing a column detection data structure for the selected rectangle which may be for example an array of Boolean values, wherein each Boolean value corresponds to a horizontal logical coordinate on the display or a character in a string of characters. This array generally represents the horizontal dimension of the table. This array is used to detect any "white space" or absence of text that extends from the top to the bottom of the selected rectangle. For each command or text string within the selection coordinates, this column detection data structure is modified in step 1806. In particular, the Boolean value for each coordinate or character in the array of Boolean values is set to true if text is present at that coordinate from any of the text content identified within the selection rectangle. The end result of step 1806 is that each occurrence of one or more contiguous false values indicates a column boundary. In step 1808, using the Y coordinates for each text item, the text for each row is identified. A string, such as an ASCII string, of cell values separated by tab characters and with a carriage return at the end for each row of the table is then built. For each text item on a given row, its X coordinate is compared with the column detection structure to determine in which column it belongs. A current count is kept of the number of tabs inserted for each row. As each cell is added, the correct number of tab characters to precede it can be calculated. This operation correctly accounts for empty cells by inserting extra tabs as needed. The tab delimited text is generated from the text commands using this operation in step 1810. From the tab delimited text, other applications readily can convert such text into tables, or spreadsheet applications can insert values into individual cells in the spreadsheet.

Providing a system as described above, in which a standard format of the visual form of data from an application is captured into a database, permits data created, used and shared over a period of time by multiple different users with multiple different computer programs that are dispersed geographically to be centrally organized, managed and accessed in a manner that is independent of the applications that create the data. The database also provides a central repository in which data is in a single format which can be searched and read for

display, printing or copying to other applications. The use of a single standard format permits the data to be viewed without the original application that viewed the data. Users also can create compound documents from multiple sources of data without a need to access the applications from which the data originated.

Having now described a few embodiments, it should be apparent to those skilled in the art that the foregoing is merely illustrative and not limiting, having been presented by way of example only. Numerous modifications may be made.

For example, the print command is used in the foregoing example because most applications at least have the ability to print a visual form of data that they manipulate and display. Other functions commonly used in applications, such as a "cut and paste" sequence also involve function calls to the operating system to translate data from one application to a different standard format in temporary storage, for example a "Clipboard" or an "Object linking and embedding (OLE) server" in Windows. This data may be retrieved from temporary storage by another application. These and other operating system functions could be used to provide access to standard format data which then may be stored in a database.

The send operation described above to transfer reports from the query interface or report previewer to another application also could be used to place data from a source application into a destination application that captures the data into a database.

Other operating systems also may be used. For example using Unix, PostScript data may be captured from an application and stored in the database. The PostScript data may be converted to a metafile format. Alternatively, a print driver could be prepared for any operating system.

These and other embodiments are within the scope of one of ordinary skill in the art and are contemplated as falling within the scope of the invention.

What is claimed is:



CLAIMS

1. A computer implemented process for capturing the visual form of information from an application executed on a computer into a database, comprising:

receiving the visual form of the data from the application;

receiving tags corresponding to the visual form of the data; and

storing the visual form of the data as a vector image and the tags into the database.

2. The computer implemented process of claim 1, wherein the database comprises:

a first table for storing tags corresponding to the visual form of the data and a reference to the visual form of the data; and

a second table for storing the visual form of the data as a vector image with a reference to the visual form of the data.

3. A database for storing the visual form of data from an application executed on a computer, comprising:

a first table for storing tags corresponding to the visual form of the data and a reference to the visual form of the data; and

a second table for storing the visual form of the data as a vector image with a reference to the visual form of the data.

4. A computer system for accessing a database in which visual form of data from applications executed on one or more computers are stored as vector images, comprising:

means for searching the database to identify one or more vector images;

means for sorting the identified vector images according to tags associated with the identified vector images; and

means for permitting a user to select one or more of the identified vector images; and

means for displaying the selected vector images.

5. The computer system of claim 4, wherein the database comprises:

a first table for storing tags corresponding to the visual form of the data and a reference to the visual form of the data; and

a second table for storing the visual form of the data as a vector image with a reference to the visual form of the data.

6. The computer system of claim 4, further comprising:

means for scaling the displayed vector images and redisplaying the scaled vector image.

7. The computer system of claim 4, further comprising:

means for searching for text in the selected vector images.

8. A computer implemented process for sending a selected portion of visual form of data from a first application to a second application, comprising the steps of:

receiving an indication of a selected area of the visual form of the data;

receiving an indication of the second application;

determining keystrokes in the second application for pasting data into the second application;

determining data representing the selected area of the visual form of the data;

transferring the determined data to a memory area from which the second application may paste data; and

sending the determined keystrokes to the second application.

9. A computer implemented process for converting a visual form of text data into character delimited text data for insertion into tabular structure, comprising:

receiving an indication of a selected area of the visual form of the data;

initializing data representing presence of text within the selected area along a horizontal extent for all rows of text;

modifying the initialized data to indicate the presence of text within the selected area along the horizontal extent for all rows of text;

for each row of text, generating character delimited data according to the text in the row and the column boundaries identified in the row.

10. A computer-implemented process for searching for text in a vector image including commands specifying text and horizontal and vertical coordinates for placement of the text, comprising:

sorting the commands in the vector image according to the vertical coordinates;

ordering the sorted commands in the vector image according to the horizontal coordinates; and

generating a list structure representing an ordering of the text from the ordered commands.

11. A service layer for controlling access to a database, comprising:

means, operative in response to a request to store data in the database, for:

encoding the data,

performing an error checking calculation on the data, and

storing the encoded data and a result of the error checking calculation in the database; and

means, operative in response to a request to read data from the database, for:

decoding the read data,

performing an error checking calculation on the read data,

comparing a result of the error checking calculation of the read data with the stored result of the error checking calculation performed on the stored data and identifying any discrepancy, and

providing the decoded read data and an indication of any discrepancy identified between the stored and read data.

12. The service layer of claim 11, wherein the database comprises:

a first table for storing tags corresponding to a visual form of data from an application and a reference to the visual form of the data; and

a second table for storing the visual form of the data as a vector image with a reference to the visual form of the data.

13. A computer system for storing a visual form of data from a plurality of different applications executed on a plurality of computers into a database accessible to a plurality of computers, comprising:

a database for storing the visual form of data with corresponding tags identifying the visual form of the data;

a processor on each of the plurality of computers executing the applications having a first input for receiving the visual form of the data from one of the plurality of different applications and a second input for receiving the tags corresponding to the visual form of the data and an output for directing the visual form of the data as a vector image and the corresponding tags into the database; and

a first application on one of the plurality of computers for accessing the tags and the visual form of the data from the database in response to queries on the database.

14. The computer system of claim 13, further comprising:

a second application for creating documents including the visual form of data retrieved from the database and for storing the created documents into the database using the processor.

15. The computer system of claim 13, wherein the database comprises:

a first table for storing tags corresponding to the visual form of the data and a reference to the visual form of the data; and

a second table for storing the visual form of the data as a vector image with a reference to the visual form of the data.

16. A computer implemented process for processing a vector image to enable scaling of text in the vector image, comprising:

identifying commands in the vector image corresponding to text; and

converting specified fonts in the identified commands to scalable fonts.

17. A computer-implemented process for creating compound documents from the visual form of data from applications executed on a computer stored in a database, comprising:

capturing the visual form of information from the applications into a database, by

receiving the visual form of the data from the application;

receiving tags corresponding to the visual form of the data; and

storing the visual form of the data as a vector image and the tags into the database;

retrieving from the database the visual form of information from the applications;

creating a compound document from the visual form of information retrieved from the database; and

capturing the visual form of the compound document into the database, by

receiving the visual form of the compound document from the application;

receiving tags corresponding to the visual form of the compound document; and

storing the visual form of the compound document as a vector image and the tags into the database.

This Sheet Has Been Replaced