



(19) **United States**

(12) **Patent Application Publication**
Heninger et al.

(10) **Pub. No.: US 2003/0140045 A1**

(43) **Pub. Date: Jul. 24, 2003**

(54) **PROVIDING A SERVER-SIDE SCRIPTING LANGUAGE AND PROGRAMMING TOOL**

(52) **U.S. Cl. 707/10; 715/513**

(76) Inventors: **Troy Heninger**, South Ogden, UT (US); **Rama Rasmussen**, Ogden, UT (US)

(57) **ABSTRACT**

Correspondence Address:
KIRTON & McCONKIE
Suite 1800
60 East South Temple
Salt Lake City, UT 84111 (US)

Server side scripting language and programming tool that includes an inventive script that is added to a normal text-based document can benefit from a dynamic document architecture. The document is processed by an interpreter to access database or other information. When a browser requests a document enhanced by the present invention, the interpreter processes the document by scanning for the inventive script commands. When a script command is encountered, the interpreter executes the script command, returns the result to the web page, and strips the executable script command from the resulting "pure-format" document. Once all executable script commands have been executed and stripped, the resulting document is in a "pure-format" form, which is sent by the server to the client computer, whose browser initiated the document request. Due to the text-based format of the present invention, the novel script commands may be added, deleted or edited from a typical HTML or Web page editor or from a word processor or text editor.

(21) Appl. No.: **10/191,361**

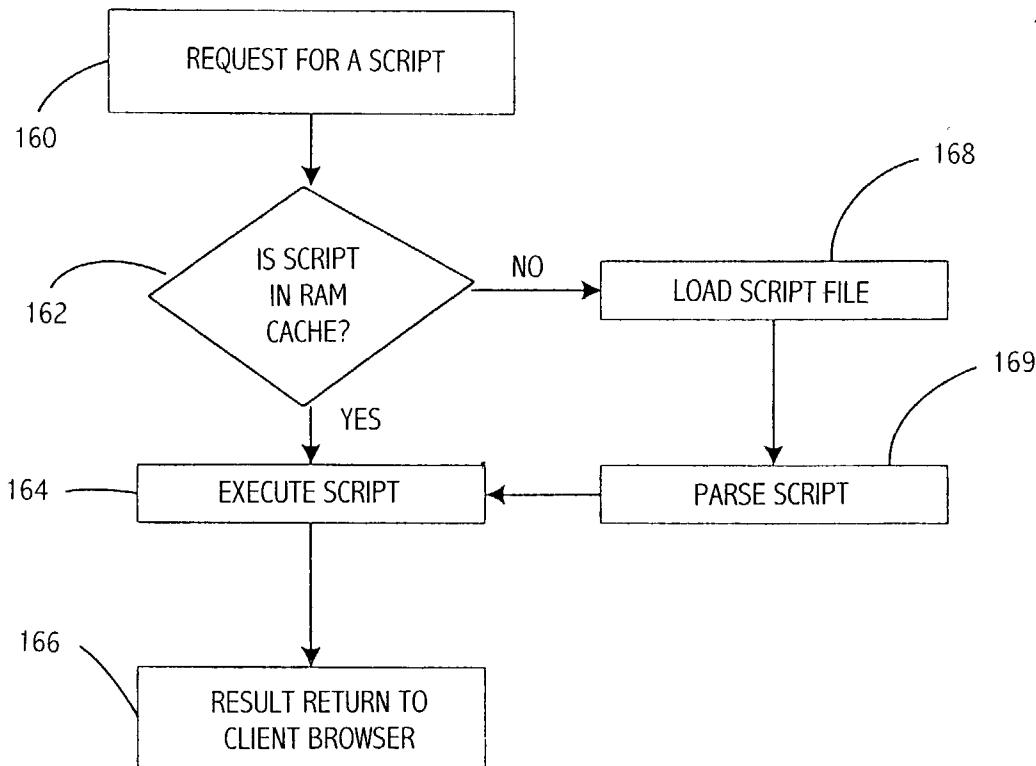
(22) Filed: **Jul. 8, 2002**

Related U.S. Application Data

(63) Continuation of application No. 09/266,357, filed on Mar. 11, 1999, now Pat. No. 6,470,349.

Publication Classification

(51) **Int. Cl.⁷ G06F 7/00; G06F 17/30**



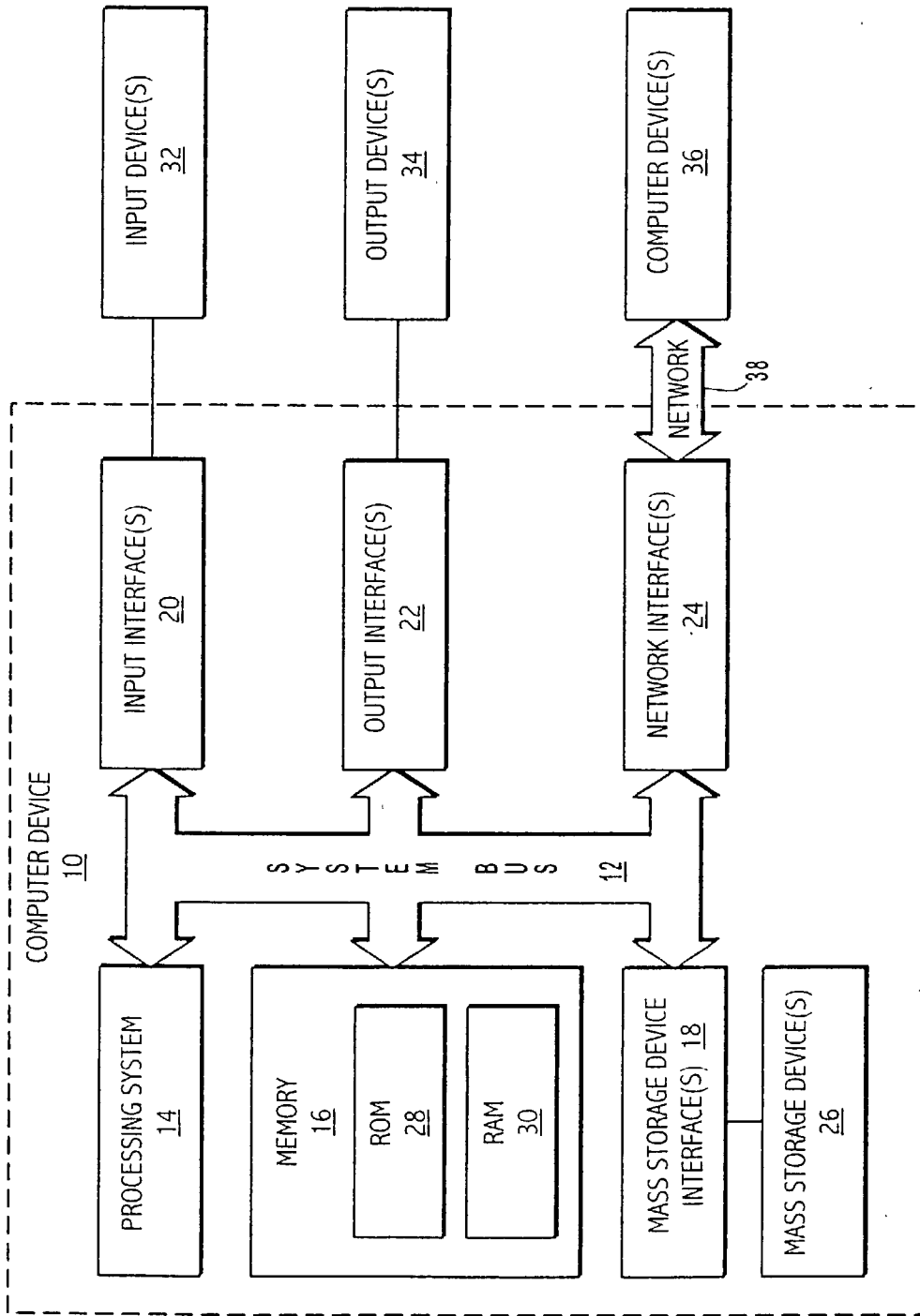


FIG. 1

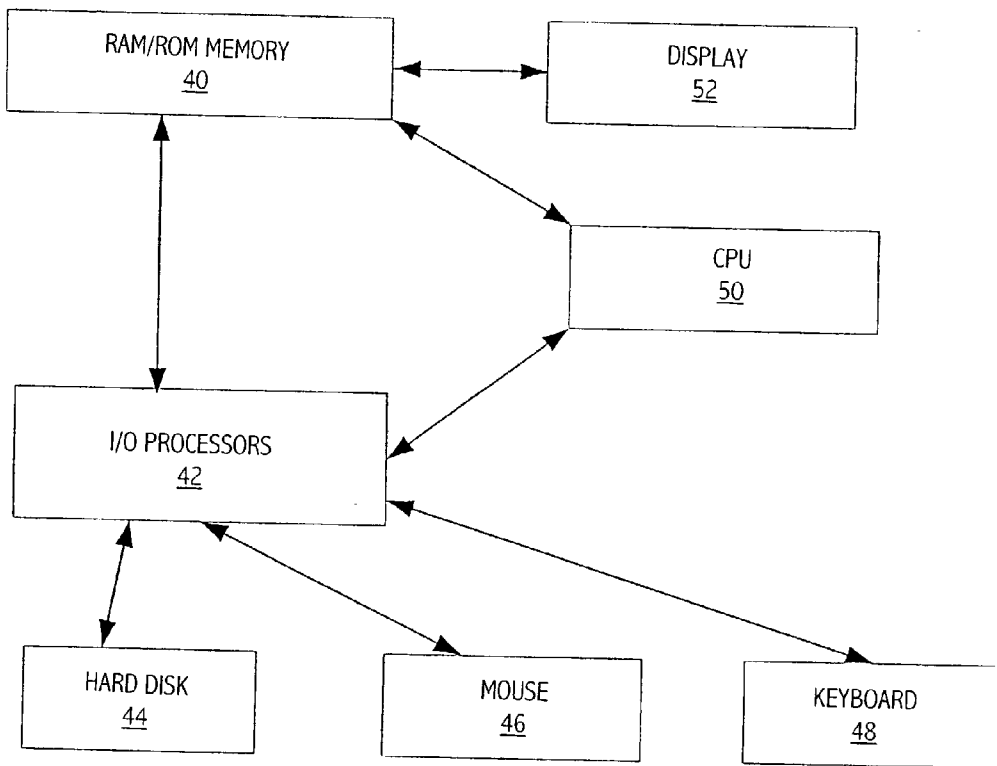


FIG. 2

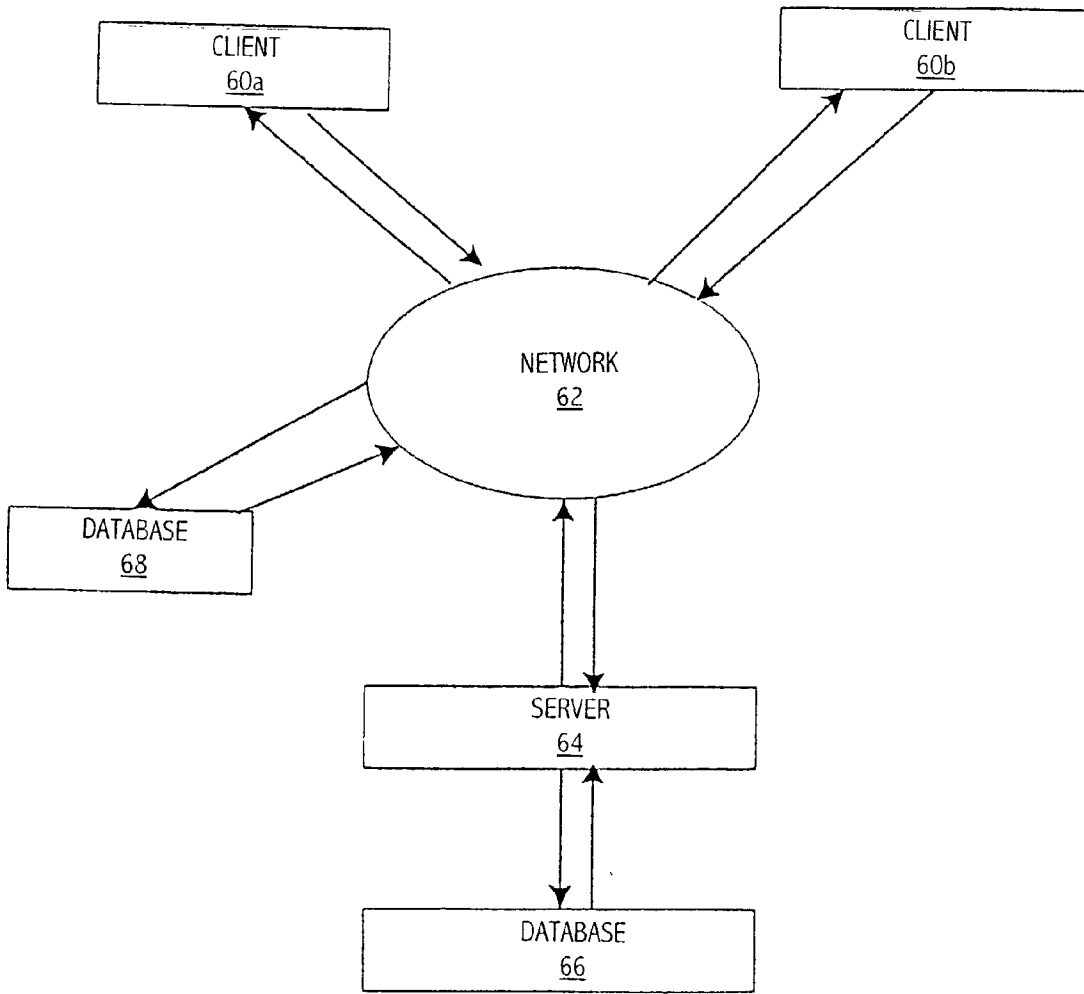


FIG. 3

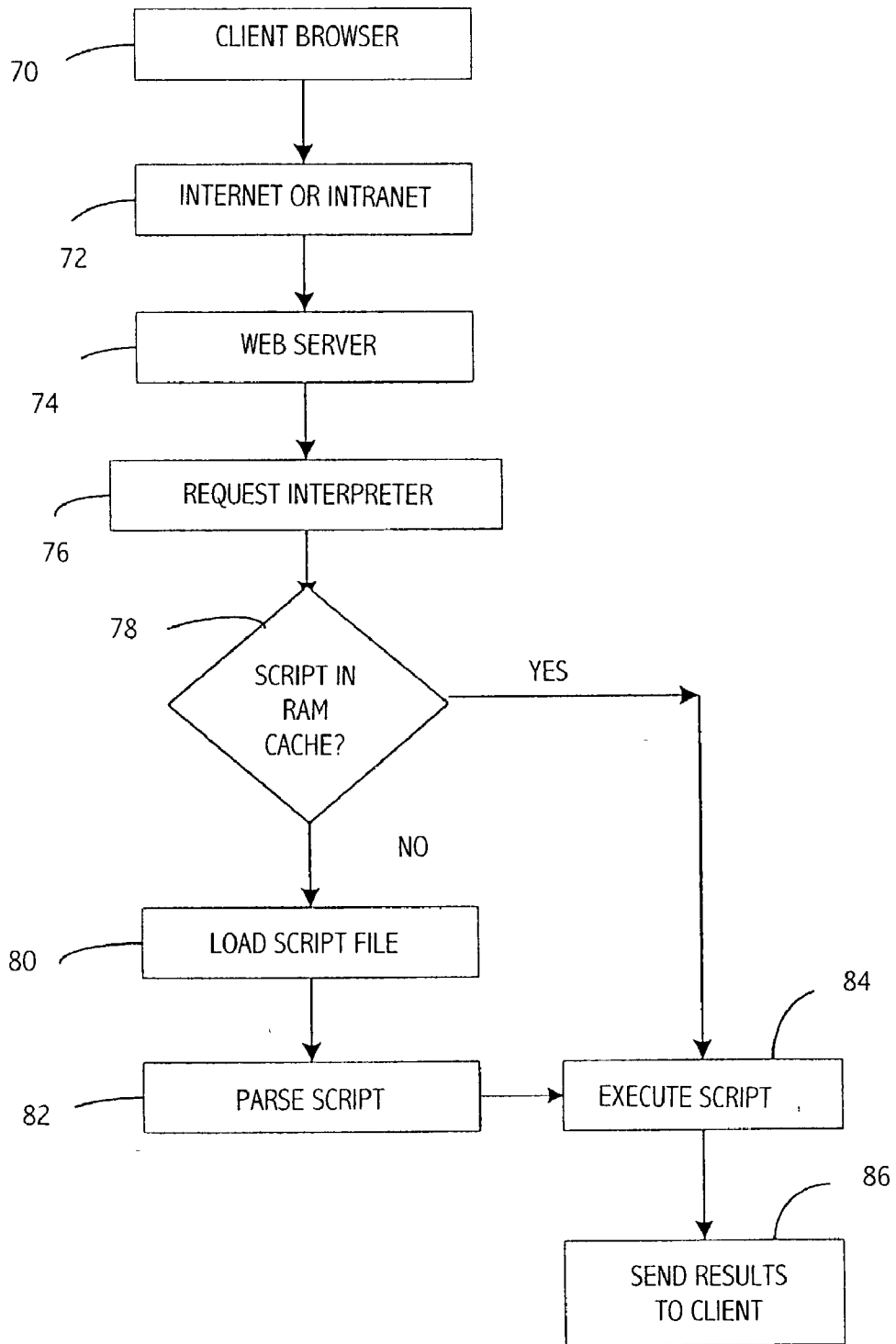


FIG. 4

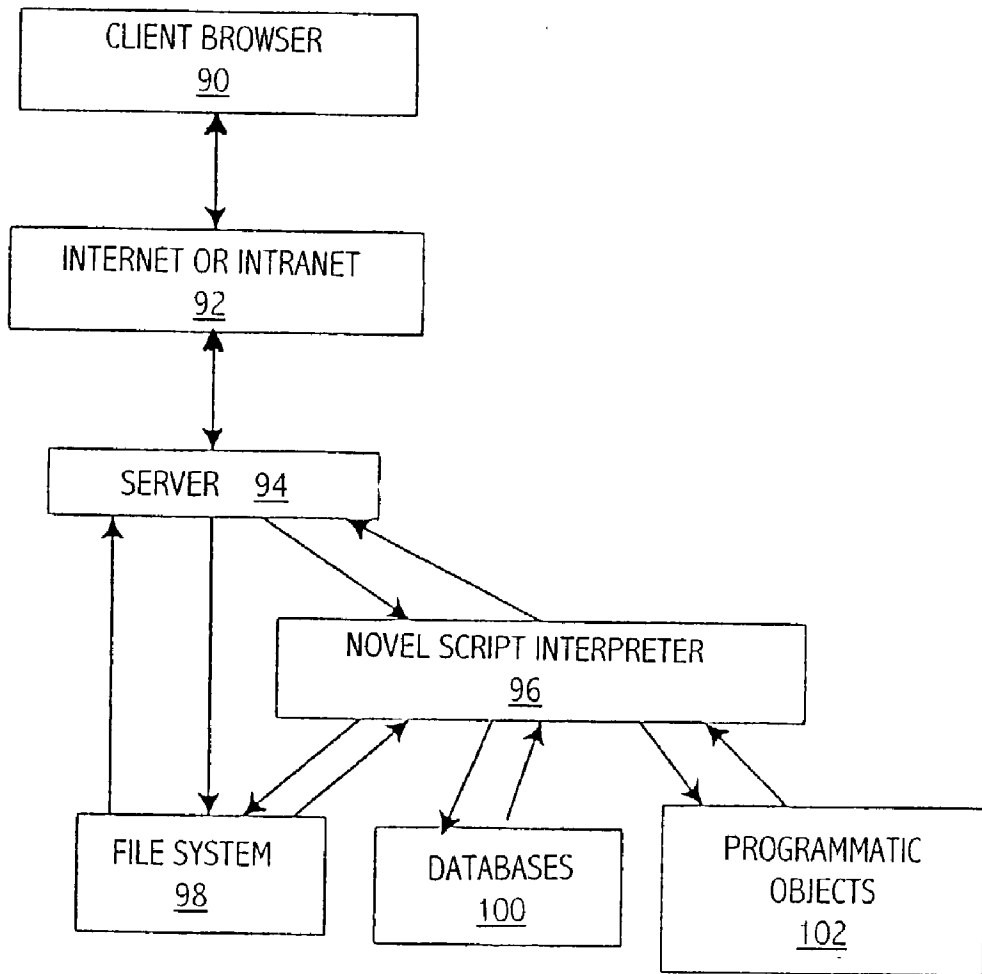


FIG. 5

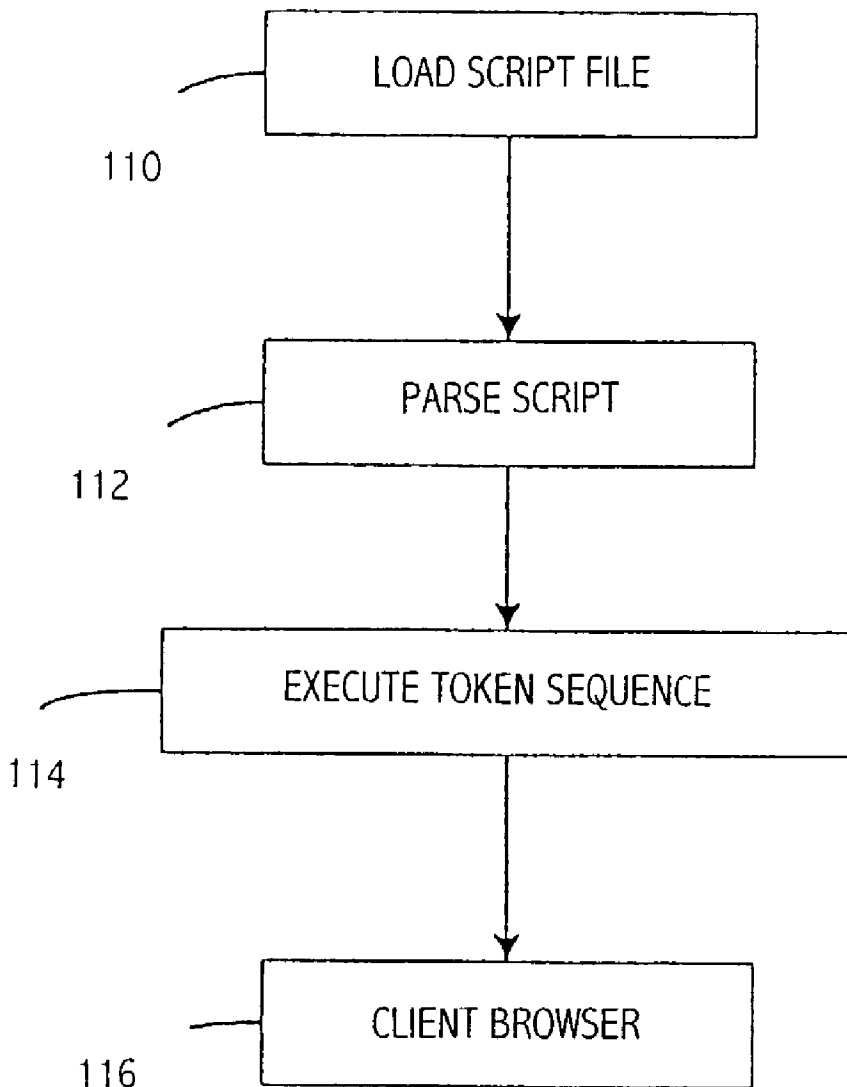


FIG. 6

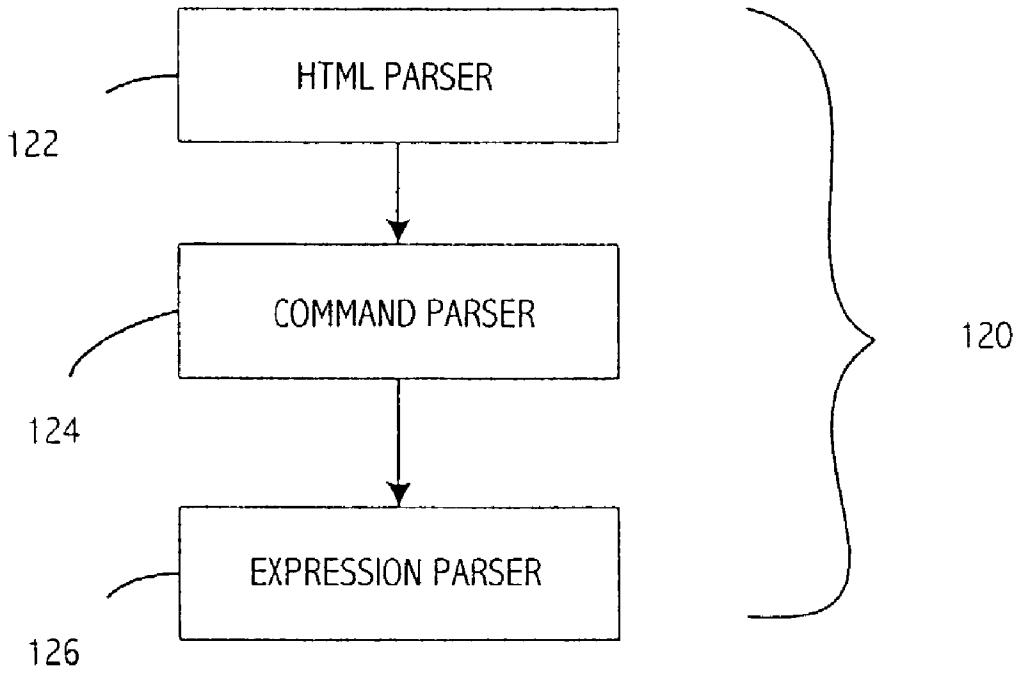


FIG. 7

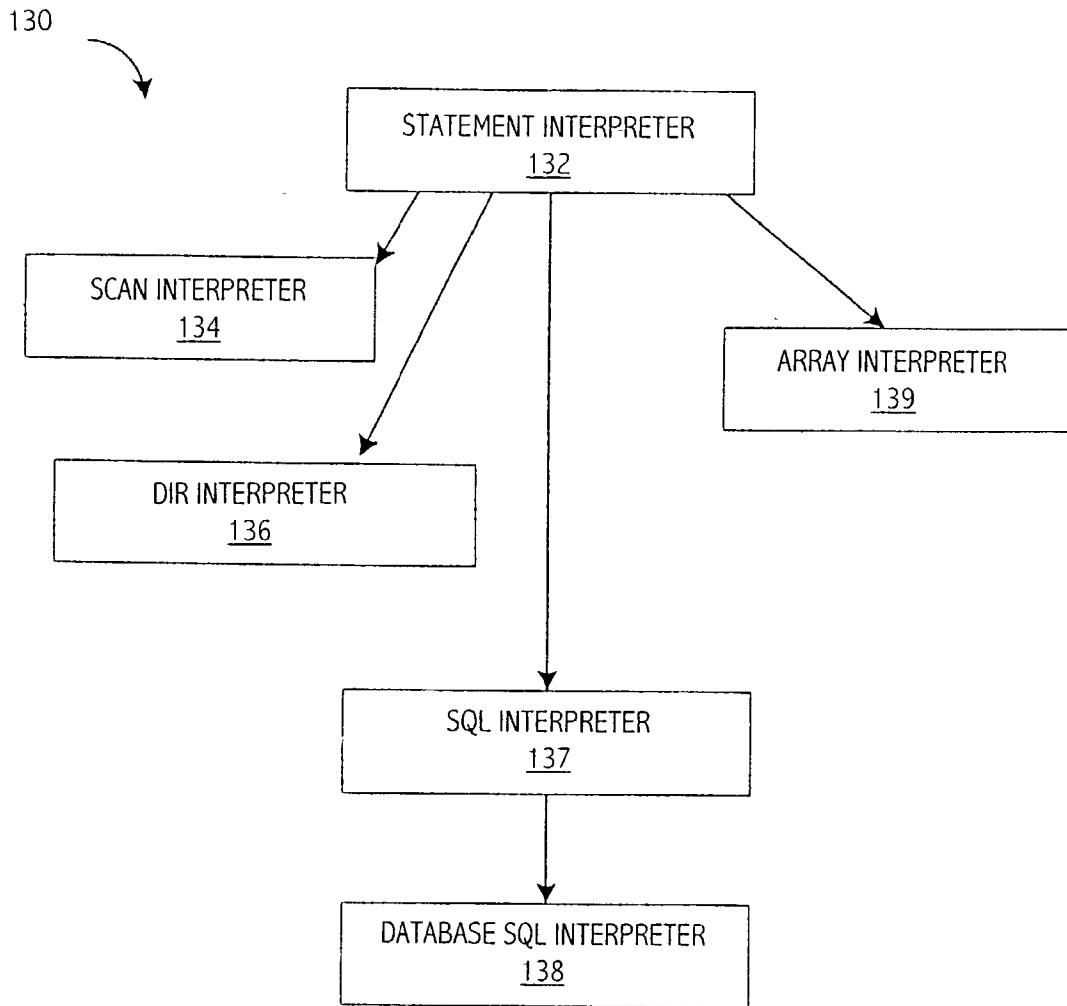


FIG. 8

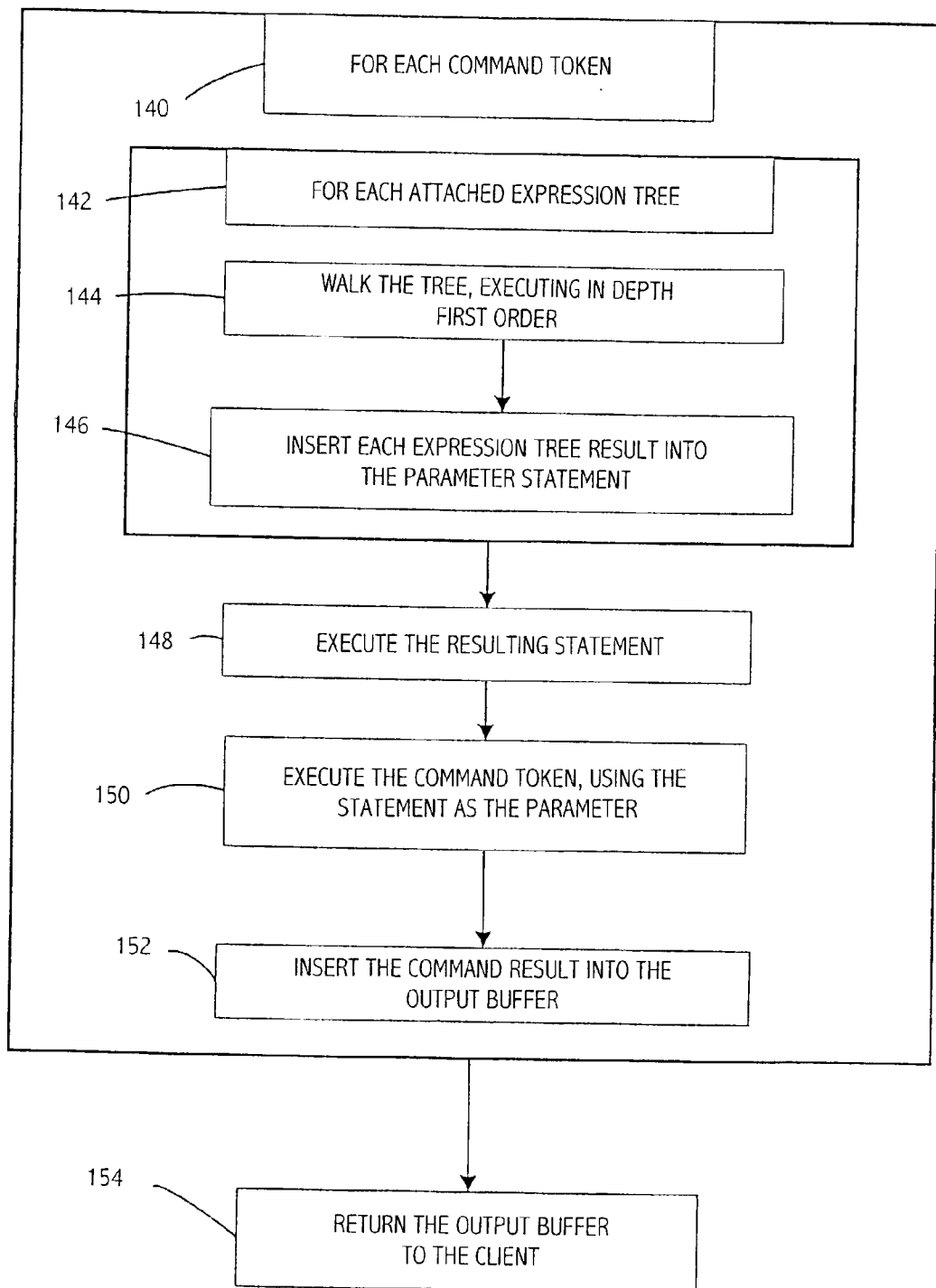


FIG. 9

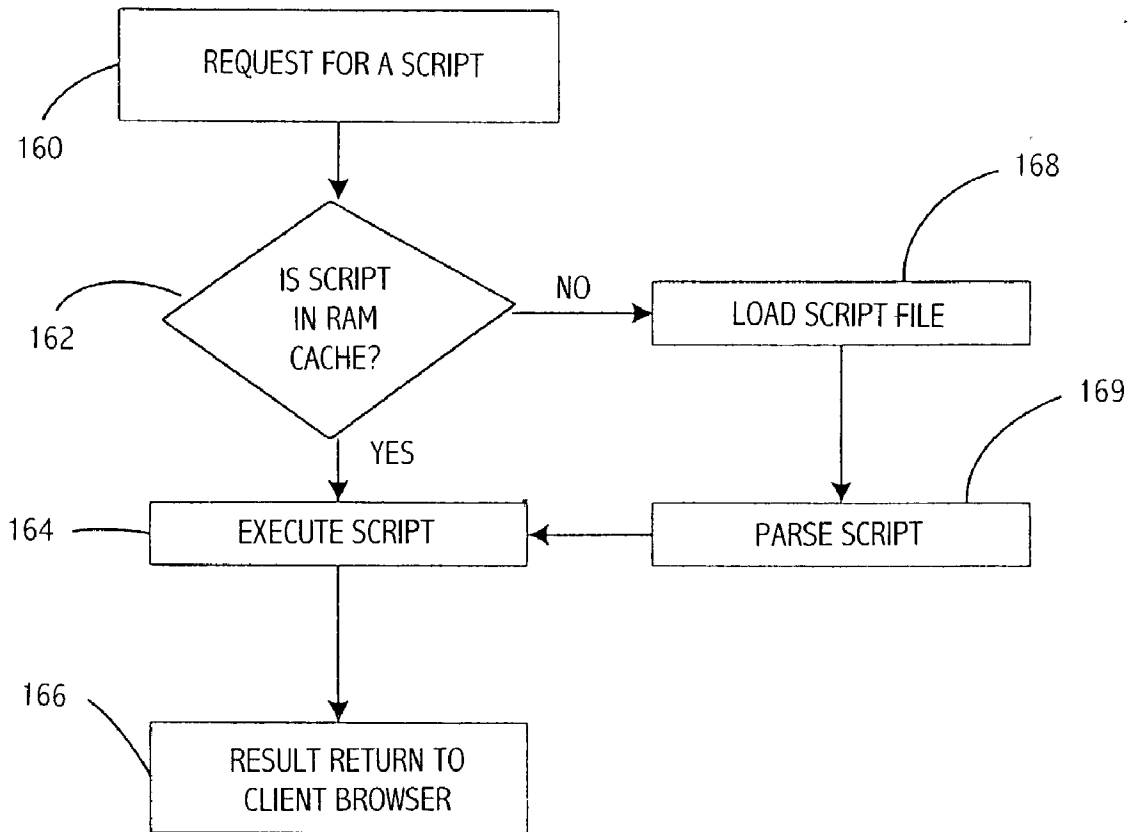


FIG. 10

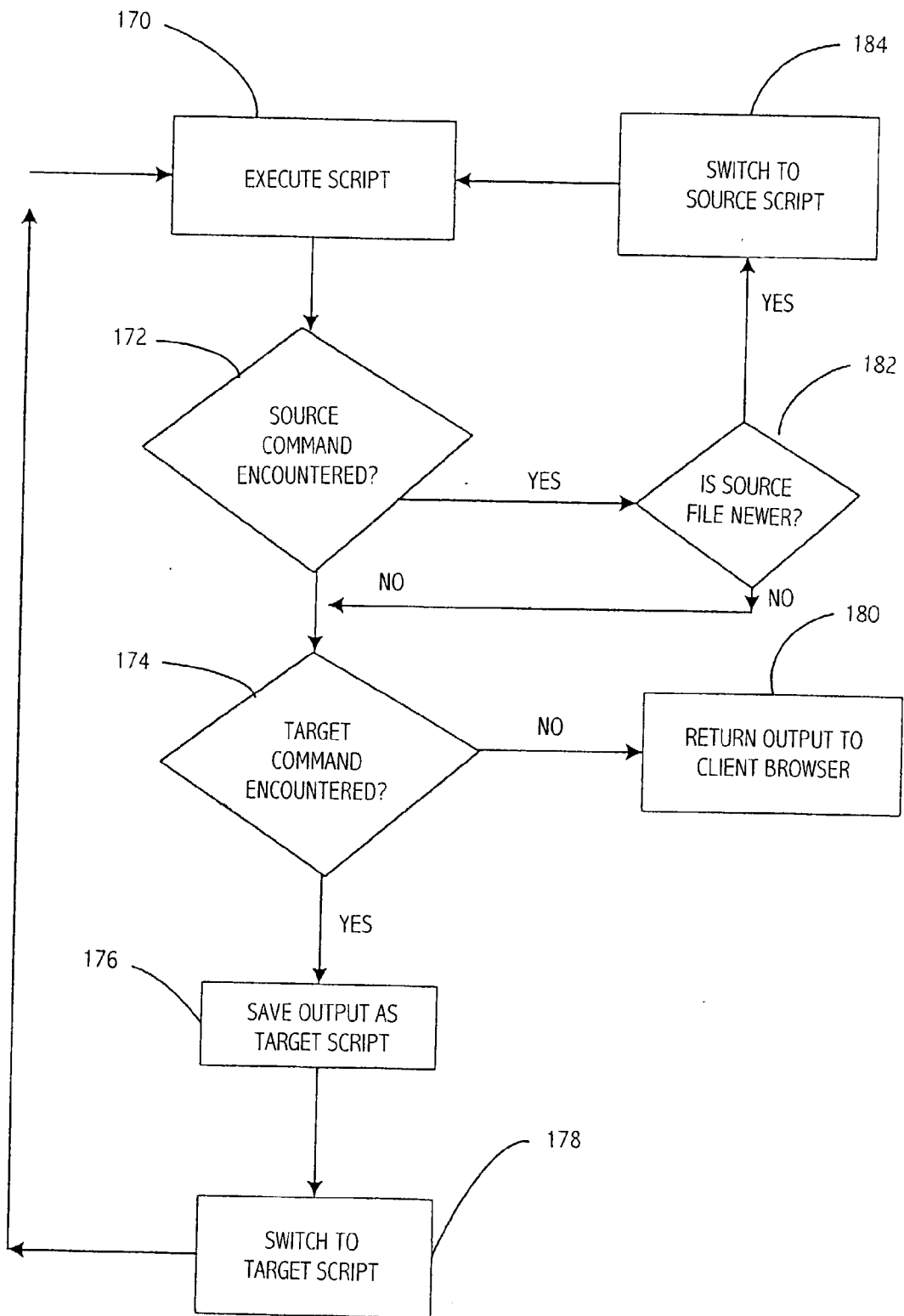


FIG. 11

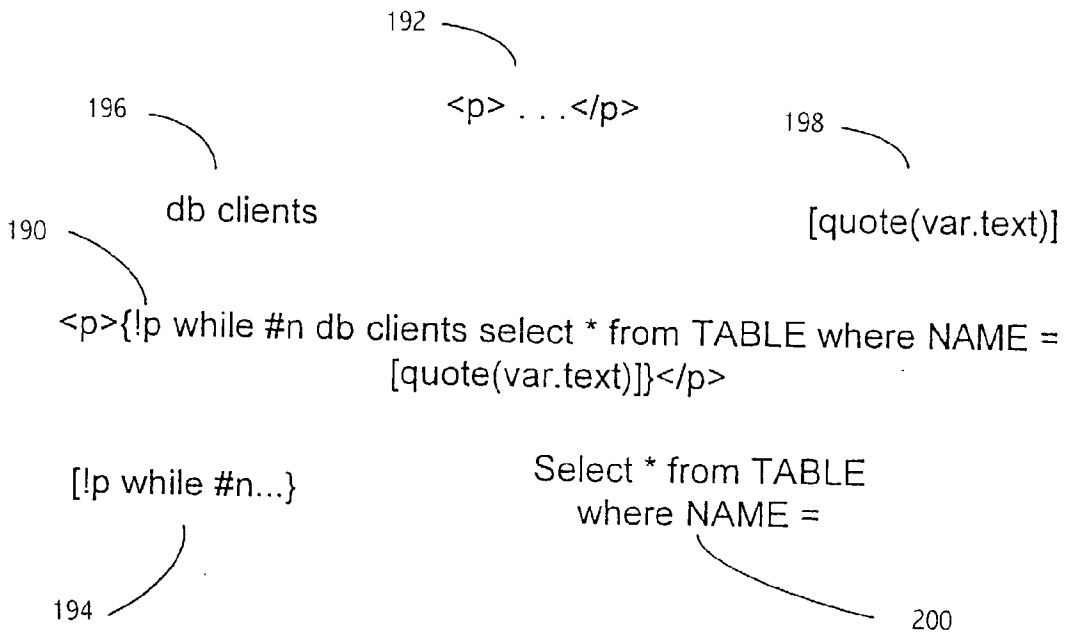


FIG. 12

PROVIDING A SERVER-SIDE SCRIPTING LANGUAGE AND PROGRAMMING TOOL

RELATED APPLICATIONS

[0001] This application is a continuation application that claims priority to U.S. patent application Ser. No. 09/266,357 filed Mar. 11, 1999, entitled SERVER-SIDE SCRIPTING LANGUAGE AND PROGRAMMING TOOL.

BACKGROUND OF THE INVENTION

[0002] 1. Field of the Invention

[0003] The present invention relates generally to the field of dynamic computer document creation. In particular, the present invention relates to a scripting language and method of use that provides for the use of script codes in a document formatted for an independent application. Implementation of the present invention is particularly well suited for creating dynamic HTML documents with database information using scripting commands.

[0004] 2. Background and Related Art

[0005] The Internet is a worldwide network of computers that is now accessible to almost any potential user with a computer having network access. Hundreds of Internet Service Providers (ISP's) have emerged to provide Internet access to millions of users who now use the Internet for personal and business communications, information retrieval, advertising, publishing, sales and document delivery.

[0006] In order to better implement many of these applications, the World Wide Web ("WWW" or "Web") was created. The Web allows access to information and documents using "hypermedia" or "hypertext" links embedded into the document. This linking system allows documents to be accessed and linked in a "non-linear" fashion, which allows a user to "jump" from one point in a document to another simply by selecting the link. It also allows jumping from one document to another whether the documents are preserved on the same computer or on another computer across the world. Documents available on the WWW include text, sound, graphics images, and even full-motion video.

[0007] Hypertext Markup Language (HTML), a specialized document formatting language, is typically used to create and format documents for viewing and linking on the Web. HTML uses special tags or codes embedded into a text document that format the document and allow linking to other documents or other locations in the same document. These links may also access files, such as sound files or graphics files that are played or displayed upon selection of a link. A document in an HTML format available on the Web is typically referred to as a "Web page" or "Web site."

[0008] The WWW also implements a unique addressing system that allocates an address or Uniform Resource Locator (URL) for each document on the Web so that Web documents may be selectively accessed, sorted and indexed.

[0009] A computer with a dedicated connection to the Internet and specialized server software that enables Web access is called a "Web server." Web documents are located on storage devices connected to Web servers. A person seeking access to a Web page uses a computer with "Web browser" software that allows access to specified URL's as well as searching and other functions.

[0010] Generally, when a user accesses a specific Web page, the user enters the URL of the desired Web page into a computer that is running browser software. The browser software, then, sends a request across the Internet to the server at the destination designated by the URL. The destination server will then send an electronic copy of the desired document to the browser computer where it can be displayed to the user. This is the typical procedure used for static Web pages that exist as unchanging files on a server storage device.

[0011] Web pages may also exist as dynamic documents that change or update themselves when conditions are met. Dynamic documents may update themselves each time they are accessed. This is achieved by allowing the server to execute a program when a given URL is accessed.

[0012] A current standard for enabling dynamic pages is the Common Gateway Interface (CGI), which allows a server to run programs that can change, update or customize the web page as it is being accessed.

[0013] Programs that reside on the server and are invoked by requests from client browser applications are referred to as "server side" programs. After these programs are run and the page is modified thereby, the newly modified "dynamic" page is sent to the browser which requested the page so it can be displayed to the user.

[0014] Another method for providing a dynamic Web page employs programs that are not operating system specific. These programs can be executed on any operating system supported by the programming language. One very popular example is the Java programming language created by Sun Microsystems, Inc. With Java, a programmer can create programs and include them in her web site where the programs will run when the site is accessed by other computers on the Web. The programs will run whether the accessing computer uses an operating system identical to that of the Web site or another operating system supported by Java. Java programs that are embedded in a site and executed by the accessing computer are called "applets." Java programs that are executed by the site server when the site is accessed by another computer are called "servlets" or more generically, "server side programs." A specialized programming language that operates exclusively with scripts interpreted by server side programs may be referred to as a "server side scripting language."

[0015] CGI and other dynamically enabled servers can allow dynamic pages to access date or time information, weather information, files with pricing or inventory data or other web pages. They may also allow access to databases stored on the same server or at another location on the Internet or a local network. This capability greatly enhances the utility of web pages and allows access to databases containing huge quantities of information.

SUMMARY OF THE INVENTION

[0016] The present invention is a server side scripting language and programming tool designed to simplify programming for web pages using databases or other dynamic information. The present invention is implemented by adding script to a normal text-based document, such as an HTML document, a spreadsheet, a word processing document or any other text-based document that can benefit from

a dynamic document architecture. The document is subsequently processed by the interpreter of the present invention to access database or other information when the document is requested by a browser or at some other time or interval specified by the programmer.

[0017] When a browser requests a document enhanced by the present invention, the interpreter processes the document by scanning for the inventive script commands. When a script command is encountered, the interpreter executes the script command, typically by retrieving or processing database information, returns the result to the web page and strips the executable script command from the resulting "pure-format" document. Once all executable script commands have been executed and stripped, the resulting document is in a "pure-format" form. In the case of a typical web page, the "pure-format" form would be a pure HTML document containing no extraneous tags or commands. This "pure-format" document is, then, sent by the server to the client computer whose browser initiated the document request.

[0018] Due to the text-based format of the present invention, the novel script commands may be added, deleted or edited from a typical HTML or Web page editor or from a word processor or text editor.

[0019] While the methods and processes of the present invention have proven to be particularly useful in the area of creating dynamic HTML documents with database information using scripting commands, those skilled in the art can appreciate that the methods and processes can be used in a variety of different applications to provide dynamic computer document creation.

[0020] These and other features and advantages of the present invention will be set forth or will become more fully apparent in the description that follows and in the appended claims. The features and advantages may be realized and obtained by means of the instruments and combinations particularly pointed out in the appended claims. Furthermore, the features and advantages of the invention may be learned by the practice of the invention or will be obvious from the description, as set forth hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0021] In order that the manner in which the above recited and other features and advantages of the present invention are obtained, a more particular description of the invention will be rendered by reference to specific embodiments thereof, which are illustrated in the appended drawings. Understanding that the drawings depict only typical embodiments of the present invention and are not, therefore, to be considered as limiting the scope of the invention, the present invention will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

[0022] **FIG. 1** illustrates a representative system that provides a suitable operating environment for use of the present invention;

[0023] **FIG. 2** illustrates a computer hardware diagram as used in the implementation of an embodiment of the present invention;

[0024] **FIG. 3** illustrates computer hardware diagram showing the interconnection of prior art client and server hardware used to implement a preferred embodiment of the present invention;

[0025] **FIG. 4** illustrates a flow chart showing a dynamic document access process of a preferred embodiment of the present invention wherein a caching process is used to speed performance;

[0026] **FIG. 5** illustrates a data flow chart showing data requests and file transfer between components of a preferred embodiment of the present invention;

[0027] **FIG. 6** illustrates a flow chart depicting the process whereby the inventive script of the preferred embodiment of the present invention is loaded, parsed, interpreted and executed and sent to the client;

[0028] **FIG. 7** illustrates a flow chart showing details of the script parsing process;

[0029] **FIG. 8** illustrates a flow chart showing details of script parsing and execution;

[0030] **FIG. 9** illustrates a flow chart showing the processing of expressions and tokens;

[0031] **FIG. 10** illustrates a flow chart showing how a caching process of the present invention is implemented;

[0032] **FIG. 11** illustrates a flow chart showing the function of TARGET and SOURCE commands in script execution; and

[0033] **FIG. 12** illustrates a script command as used in a HTML document in accordance with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0034] The present invention relates to generally to the field of dynamic computer document creation. In particular, the present invention relates to systems and methods for providing a scripting language that uses script codes in a document formatted for an independent application. Implementation of the present invention is particularly well suited for creating dynamic HTML documents with database information using scripting commands.

[0035] The following disclosure of the present invention is grouped into two subheadings, namely "Exemplary Operating Environment" and "Providing a Scripting Language." The utilization of the subheadings is for convenience of the reader only and is not to be construed as limiting in any sense.

Exemplary Operating Environment

[0036] Embodiments of the present invention embrace the utilization of one or more computer device to provide the systems and methods discussed herein. Accordingly, **FIG. 1** and the corresponding discussion are intended to provide a general description of a suitable operating environment in which the invention may be implemented. One skilled in the art will appreciate that the invention may be practiced by one or more computing devices and in a variety of system configurations, including in a networked configuration.

[0037] Embodiments of the present invention embrace one or more computer readable media, wherein each medium may be configured to include or includes thereon data or computer executable instructions for manipulating data. The computer executable instructions include data structures, objects, programs, routines, or other program modules that

may be accessed by a processing system, such as one associated with a general-purpose computer capable of performing various different functions or one associated with a special-purpose computer capable of performing a limited number of functions. Computer executable instructions cause the processing system to perform a particular function or group of functions and are examples of program code means for implementing steps for methods disclosed herein. Furthermore, a particular sequence of the executable instructions provides an example of corresponding acts that may be used to implement such steps. Examples of computer readable media include random-access memory ("RAM"), read-only memory ("ROM"), programmable read-only memory ("PROM"), erasable programmable read-only memory ("EPROM"), electrically erasable programmable read-only memory ("EEPROM"), compact disk read-only memory ("CD-ROM"), or any other device or component that is capable of providing data or executable instructions that may be accessed by a processing system.

[0038] With reference to FIG. 1, a representative system for implementing the invention includes computer device 10, which may be a general-purpose or special-purpose computer. For example, computer device 10 may be a personal computer, a notebook computer, a personal digital assistant ("PDA") or other hand-held device, a workstation, a minicomputer, a mainframe, a supercomputer, a multi-processor system, a network computer, a processor-based consumer electronic device, or the like.

[0039] Computer device 10 includes system bus 12, which may be configured to connect various components thereof and enables data to be exchanged between two or more components. System bus 12 may include one of a variety of bus structures including a memory bus or memory controller, a peripheral bus, or a local bus that uses any of a variety of bus architectures. Typical components connected by system bus 12 include processing system 14 and memory 16. Other components may include one or more mass storage device interfaces 18, input interfaces 20, output interfaces 22, and/or network interfaces 24, each of which will be discussed below.

[0040] Processing system 14 includes one or more processors, such as a central processor and optionally one or more other processors designed to perform a particular function or task. It is typically processing system 14 that executes the instructions provided on computer readable media, such as on memory 16, a magnetic hard disk, a removable magnetic disk, a magnetic cassette, an optical disk, or from a communication connection, which may also be viewed as a computer readable medium.

[0041] Memory 16 includes one or more computer readable media that may be configured to include or include thereon data or instructions for manipulating data, and may be accessed by processing system 14 through system bus 12. Memory 16 may include, for example, ROM 28, used to permanently store information, and/or RAM 30, used to temporarily store information. ROM 28 may include a basic input/output system ("BIOS") having one or more routines that are used to establish communication, such as during start-up of computer device 10. RAM 30 may include one or more program modules, such as one or more operating systems, application programs, and/or program data.

[0042] One or more mass storage device interfaces 18 may be used to connect one or more mass storage devices 26 to

system bus 12. The mass storage devices 26 may be incorporated into or may be peripheral to computer device 10 and allow computer device 10 to retain large amounts of data. Optionally, one or more of the mass storage devices 26 may be removable from computer device 10. Examples of mass storage devices include hard disk drives, magnetic disk drives, tape drives and optical disk drives. A mass storage device 26 may read from and/or write to a magnetic hard disk, a removable magnetic disk, a magnetic cassette, an optical disk, or another computer readable medium. Mass storage devices 26 and their corresponding computer readable media provide nonvolatile storage of data and/or executable instructions that may include one or more program modules such as an operating system, one or more application programs, other program modules, or program data. Such executable instructions are examples of program code means for implementing steps for methods disclosed herein.

[0043] One or more input interfaces 20 may be employed to enable a user to enter data and/or instructions to computer device 10 through one or more corresponding input devices 32. Examples of such input devices include a keyboard and alternate input devices, such as a mouse, trackball, light pen, stylus, or other pointing device, a microphone, a joystick, a game pad, a satellite dish, a scanner, a camcorder, a digital camera, and the like. Similarly, examples of input interfaces 20 that may be used to connect the input devices 32 to the system bus 12 include a serial port, a parallel port, a game port, a universal serial bus ("USB"), a firewire (IEEE 1394), or another interface.

[0044] One or more output interfaces 22 may be employed to connect one or more corresponding output devices 34 to system bus 12. Examples of output devices include a monitor or display screen, a speaker, a printer, and the like. A particular output device 34 may be integrated with or peripheral to computer device 10. Examples of output interfaces include a video adapter, an audio adapter, a parallel port, and the like.

[0045] One or more network interfaces 24 enable computer device 10 to exchange information with one or more other local or remote computer devices, illustrated as computer devices 36, via a network 38 that may include hard-wired and/or wireless links. Examples of network interfaces include a network adapter for connection to a local area network ("LAN") or a modem, wireless link, or other adapter for connection to a wide area network ("WAN"), such as the Internet. The network interface 24 may be incorporated with or peripheral to computer device 10. In a networked system, accessible program modules or portions thereof may be stored in a remote memory storage device. Furthermore, in a networked system computer device 10 may participate in a distributed computing environment, where functions or tasks are performed by a plurality of networked computer devices.

Providing a Scripting Language

[0046] As provided above, embodiments of the present invention relate to a scripting language and method of use that provides for the use of script codes in a document formatted for an independent application. Implementation of the present invention is particularly well suited for creating dynamic HTML documents with database information using

scripting commands. In a preferred embodiment, a server side scripting language and programming tool greatly simplifies programming of many types of Web pages, including Web database applications.

[0047] A first step in using a preferred embodiment of the present invention is to create an initial HTML document with the desired structure, layout and format defined by HTML code. After the initial HTML document is complete, the inventive commands of the scripting language of the preferred embodiment of the present invention are added to the initial HTML document to transform the initial HTML document into a mixed-format scripted document. This mixed-format scripted document contains the HTML code of the initial HTML document combined with the inventive script commands of the preferred embodiment of the present invention. These script commands, which are input by a programmer, contain commands that add functionality to the Web page being created. Some of these commands may implement procedures that are executed when the resulting Web page is accessed thereby creating a dynamic Web page.

[0048] The currently preferred embodiment of the present invention utilizes a Java servlet to interpret the novel script commands. Accordingly, the server supports servlets for this embodiment. Servlets may be supported natively or through the use of plug-in programs that are common in the industry. The majority of these plug-in programs utilize a version of Java that is installed on the server. The Java Runtime Environment (JRE) or the Java Development Kit (JDK) may be installed on the server to enable servlet support. The Java environment also provides compatibility with virtually all established operating systems in use today.

[0049] The presently preferred embodiment of the present invention is entirely contained within one or more files stored in a standard file system, thereby eliminating the need for a bulky, cumbersome database for program operation and functionality. This structure improves performance and eliminates the need for licensing of third party database technology.

[0050] This implementation is quickly and easily employed for many applications, however, if more speed is required or if speed takes precedence over ease of implementation, another embodiment may be used that employs a server add-on using the Common Gateway Interface (CGI) standard or a server plug-in using the Internet Server Application Programming Interface—Dynamic Load Library (ISAPI—DLL) standard.

[0051] When the presently preferred embodiment of the present invention is used to provide database connectivity to Web pages or other documents, a Java Database Connectivity (JDBC) or Open Database Connectivity (ODBC) driver is utilized to communicate with the database. Currently, JDBC is the preferred driver as it provides better performance with the preferred software embodiment.

[0052] A preferred embodiment of the present invention generally utilizes one or more computer devices or computers in a network environment. As indicated above, these computers, as illustrated in FIGS. 1 & 2, may be any type of computer device, including mainframes, mini-computers, micro-computers or other variations, however, the most commonly used variations will employ a central processing unit (CPU) 50, with RAM and/or ROM memory 40, a

display device 52 such as a cathode ray tube (CRT), one or more input/output processors 42, a hard disk 44 or other mass storage device, a pointing device, such as a mouse 46, an input device such as a keyboard 48 and various other peripherals or other devices.

[0053] A simple networking environment is illustrated in FIG. 3 where one or more client computers 60 are connected to a computer network 62. This computer network 62 may be a localized network, an intranet, or may utilize a wide-area network structure or the Internet to communicate with a server 64. Server 64 may communicate with a database 66 which can be physically located on the server hardware itself or elsewhere. In some embodiments, server 64 may communicate directly with database 66 or may employ the network 62 or Internet communications capability to access the database, as illustrated by database 68.

[0054] In a preferred embodiment of the present invention, as shown in FIG. 4, a client computer running web browser software 70 is connected to the Internet or an intranet 72. The client web browser initiates a request for a specific Web page represented by a URL address. The request is sent over the Internet or intranet 72 to the web server 74 where the URL is located. The web server 74 finds the document associated with the requested URL and, in a preferred embodiment, checks the file extension of the document that indicates whether inventive script commands of the present invention are contained therein. If the inventive script commands are not present in the document, the document is immediately sent to the client web browser in its current form.

[0055] If the inventive script commands are present in the document, the document is sent to the request interpreter 76 of the present invention. At this point, the request interpreter 76 checks to see if the script has already been loaded and parsed making it ready for execution. This may have taken place during a prior request for the Web page. If so, the script will be held in a RAM cache or otherwise stored in an executable form 78 and will be immediately executed 84 when the page is requested. If the file has not been previously accessed, the script will have to be loaded 80 and parsed 82 before execution 84. The results are then sent to the client at 86.

[0056] The data flow characteristics of the preferred embodiment of the present invention is shown in FIG. 5 where a client web browser 90 is again shown making a request over the Internet or an intranet 92 for a specific web page. The web server 94 receives the request and, when the file extension indicates that inventive script commands are not present in the document, accesses the file system 98 to retrieve the requested document and deliver it to the requesting browser 90. When the inventive script commands of the present invention are present in the document, the request is passed to the novel script interpreter 96. The script interpreter 96 interprets the inventive script commands in the document and, depending on the nature of the script commands, may access files in the file system 98, access one or more databases 100 or execute one or more programmatic objects 102. A detailed list of the inventive script commands of the present invention and the actions they invoke is given below.

[0057] The script processing method of the preferred embodiment of the present invention begins by loading the

script file **110**. In the Web application of the preferred embodiment, this file will be an HTML document that contains, in addition to text and HTML code, novel script commands. However, in other embodiments of the present invention, the novel script commands may be used to enhance other types of files. By way of example and not by limitation, these files may be spreadsheet files, word processor files or graphics files or combinations of file formats such as graphics or other files embedded in HTML documents. A script file may be any file that contains the novel script commands of the present invention.

[**0058**] Once the script file is loaded, the interpreter of the preferred embodiment of the present invention parses the script **112** thereby creating sequentially ordered tokens for each element of the file. These elements comprise HTML tags, text, inventive script commands of the present invention, and expressions used by these inventive commands such as database or file locations. After the file elements have been converted to executable tokens they are assembled into an executable token sequence that will be executed immediately and cached so it may be re-executed each time the page is requested in the future. Execution of the token sequence **114** creates a new "pure-format" or "pure-HTML" document that is sent to the client browser that requested the document **16**. Accordingly, a dynamic Web page is created as the page is updated or recreated each time the page is requested.

[**0059**] The parsing procedure of the preferred embodiment of the present invention is shown in more detail in **FIGS. 7 and 8**. Parsing takes place at two distinct times in the process of the preferred embodiment, when the script is initially loaded **120** (**FIG. 7**) and when the file is executed **130** (**FIG. 8**).

[**0060**] When a script file is initially loaded **120**, the elements of the file are parsed by a series of parsers that recognize and process individual types of commands and their related expressions. The HTML parser **122** distinguishes between HTML tags and inventive script commands in the file and creates tokens for contiguous HTML blocks. The Command parser **124** recognizes the inventive script commands of the present invention and creates tokens for each of these commands. When an inventive script command contains one or more expressions, the expression portion of the inventive script command is sent to the expression parser **126** which creates a token for the expression which is passed back to the command parser **124** and incorporated into the token for the inventive script command.

[**0061**] A second parsing process occurs at the time the token file is executed **130**. This occurs each time a statement is encountered while executing a script. At this point, the statement parser **132** determines the type of statement and sends the statement to the interpreter appropriate for that statement.

[**0062**] When a scan statement is encountered by the statement parser, the statement is sent to the Scan interpreter **134** which interprets the statement by extracting designated words from strings as indicated in the statement.

[**0063**] When the statement parser encounters a directory statement, the statement is sent to the Dir interpreter **136** which will interpret the statement and enumerate the files within a directory.

[**0064**] The statement parser will, likewise, find array statements and send them to the Array interpreter **139** which will interpret those statements by iterating through the designated arrays.

[**0065**] SQL and database related commands are sent by the statement parser **132** to the SQL interpreter **137**, which will directly interpret a database-switching modifier. Other database commands or the remainder of a statement containing a database switching command will be forwarded by the SQL interpreter **137** to a Database SQL interpreter **138** which is external to the preferred embodiment of the present invention and typically resides in a database driver or engine program.

[**0066**] During the script execution process, as detailed in **FIG. 9**, each command token is processed sequentially. With reference to step **140**, if expression trees are attached to command tokens, they are also processed in the order encountered. Expression trees **142** follow an expression hierarchy or tree structure. When the result of an expression is dependent upon another, the script execution process of the preferred embodiment of the present invention executes the expressions in order **144** starting with the outermost expressions on the expression tree. The expression results are, then, inserted into the expressions of the next order until all expressions have been processed and the main parameter statement is ready to execute **146**. This statement is subsequently parsed out **148** by the statement parser **132** (**FIG. 8**) to the appropriate interpreter that executes the command token **150** thereby returning a result that is sent to an output buffer **152** where it is combined with the results of other statements to form the resulting dynamic Web page that is sent to the client **154**.

[**0067**] In order to increase performance, the preferred embodiment of the present invention utilizes caching techniques to increase response speed. A RAM cache technique is used to speed response time when a client requests a dynamic Web page. A file cache may also be used to increase performance.

[**0068**] When an inventive script document is requested **160**, as shown in **FIG. 10**, the server checks to see if the script is stored in RAM cache **162**. If so, the script is immediately executed **164** and the resulting HTML document is sent to the client browser **166**. If the script is not stored in RAM Cache, the script file is loaded **168**, and parsed **169** before execution **164** and, finally, sent to the client browser **166**.

[**0069**] A programmer may use the Source command or Target command to control program execution and caching and relate script files. Using the inventive scripting language of the present invention, a script file may contain script commands which, when executed, create a new script file. The file from which the new file was created is referred to as a source file. The Source command initiates a comparison between the current file and an associated source file from which the current file was created. When a Source command is encountered during execution, the creation date of the current file is compared with the creation date of the associated source file. The file with the most recent date, the newer file, is selected for execution.

[**0070**] When a Target command is encountered during script execution, the output of the execution process is written to a target file that may be another script file that can be subsequently executed.

[0071] The operation of the Target and Source commands may be more fully understood by reference to FIG. 11. During the execution of a script file 170, if a source command is encountered 172, a comparison is made to determine whether the source or current file is newer 182. If the source file is newer, the execution process switches to the source file 184 and executes the source file. If the current file is newer, execution of the current file proceeds. The Source command can, therefore, be used to ensure that the most current version of a file is used.

[0072] If a target command is not encountered 174 during execution of a script file, the output of the script execution will be sent to a client browser 180 or other source which initiated the scripted document request. When a Target command is encountered the output of the script file execution process will be written to a target file rather than being sent to the client 176. The execution process will then switch to the target script file 178 and execute the target file. The Target command allows intermediate files, which may be more easily edited, to be created. It also allows file execution to become an iterative process.

[0073] In FIG. 12, the syntax of the scripting language of the preferred embodiment of the present invention is illustrated. An example line of code is shown 190 with various tags and commands therein. Within the code are some HTML tags 192 which will be differentiated and processed by the HTML parser 122 (FIG. 7). Script command syntax 194 is parsed and processed by the command parser 127 (FIG. 7). Database switching syntax 196 within the code line is parsed and sent to the SQL interpreter 137 (FIG. 8) for processing. When found in the code, expression syntax 198 is parsed and sent to the expression parser for processing. When database access and manipulation is performed, SQL syntax 200 will be present. This will be parsed out and sent to the database SQL interpreter 138 (FIG. 8) for processing.

[0074] In order to more fully demonstrate the capabilities of the preferred embodiment of the present invention, the following list of program properties, commands and functions is presented along with an explanation of each. This listing is offered as an example of the inventive commands of the preferred embodiment of the present invention and is not to be construed as a limitation to the scope of the claims of this patent.

[0075] 0.1 Standards

[0076] The format of this listing is as follows: syntax is in italics, and commands are bolded within paragraph descriptions. Also example scripts use Courier Font.

[0077] 0.2 Introduction

[0078] The server-side scripting language and programming tool of the present invention is called BrowZ Script. Its purpose is to greatly simplify programming for many types of web database applications. To use it, create normal HTML pages, then place special tags containing BrowZ Script commands in the pages. These tags are replaced dynamically with the result of the command execution when the script is requested by a browser and executed by the server. Several of the commands execute SQL statements against a database, and others retrieve data from the resulting rows and columns. BrowZ Script takes full advantage of the power of SQL and relies heavily on the resulting data.

[0079] BrowZ Script also supports the retrieving of data from HTML page forms as well as arguments passed after a question mark '?' at the end of an Internet address or URL.

[0080] BrowZ Script has extra features for ease of programming, such as allowing you to use your favorite web page designer and editor for adding and editing of your script tags. Some knowledge of HTML is needed to take advantage of this feature.

[0081] Installation

[0082] 1.1 Server

[0083] BrowZ Script is implemented as a Java servlet. So, your web server must support servlets, either natively or by use of a server plug-in or CGI. If your server does not have built-in servlet support you must install one of several server plug-ins. Most of these servlet add-ons are free or inexpensive. Most of these plug-ins, however, will also require that you have a version of Java installed on your server. This can be freely downloaded from Sun's Java web site, www.java-soft.com. You'll need to download and install the JRE (Java Runtime Environment) or the JDK (Java Development Kit).

[0084] BrowZ Script is implemented in Java therefore you can run it on any operating system platform for which the Java runtime is available, which includes just about all of the modern ones.

[0085] 1.2 Files

[0086] BrowZ Script is entirely contained within the BZScript.jar file. You'll need to configure your server (or the server plug-in) by adding this file to its Java CLASSPATH. Some servers may work if you simply copy this file into the Servlets folder. Check with your server or plug-in for proper installation of servlets.

[0087] 1.3 Database

[0088] BrowZ Script requires a JDBC (Java DataBase Connectivity) or ODBC (Open DataBase Connectivity) driver to be able to communicate with your database. It is usually fastest if you have a JDBC driver for your particular database, but an ODBC driver will work fine if you do not. If you are using JDBC you'll need to know the JDBC URL to your database. Look in your driver's user manual for the proper URL to use for each of your databases. If you choose to use ODBC, you must set up a DSN (Data Source Name) for each database you wish to communicate with. The JDBC URL you will use will be as follows: "odbc:your_dsn".

[0089] 2.0 Properties Files

[0090] These files give information to BrowZ Script concerning all of the files in a directory and the databases with which they plan to connect. For every folder in which BrowZ script is going to process files there must be a properties file called "BZScript.properties". This is a plain text file usually containing four definitions, similar to the following:

[0091] driver=sun.jdbc.odbc.JdbcOdbcDriver

[0092] database=odbc:MyDSN

[0093] username=MyUser

[0094] password=MyPassword

[0095] The driver parameter defines which Java driver to load. In this case, we are loading the built-in ODBC driver. If you use another driver you would place its class name here. Driver defaults to sun.jdbc.odbc.JdbcOdbcDriver.

[0096] The database parameter defines the JDBC URL to the database. This is dependent on the driver you use and the database you are connecting to. Since we used the ODBC driver we specified an ODBC URL containing the DSN.

[0097] The username and password parameters are used when your database requires a user name and password to access the database. You may include a space character, if needed, within the value, but do not surround it by quotation marks, as you might think. Also, do not put spaces or tabs on either side of the equal sign '='. The name, the equals sign, and the value must be run together with no white space.

[0098] BrowZ script is not limited to only one database as described above. In order to support connections with more than one database the contents of the BZScript.properties file has to be changed a little. Let's start with an example and then explain the differences:

```
[0099] databases=customers, finance
[0100] customers.driver=sun.jdbc.odbc.JdbcOdbcDriver
[0101] customers.database=odbc:CLIENTS
[0102] finance.driver=sun.jdbc.odbc.JdbcOdbcDriver
[0103] finance.database=odbc:FINANCE
[0104] finance.username=MyUser
[0105] finance.password=MyPassword
```

[0106] First notice the databases parameter. This is where you list all of the databases with which you wish to connect. All other parameters have the database name and a period before them, specifying the database to which they belong. Unless otherwise specified, BrowZ Scripts assume a connection with only one database. This default database will always be the first one listed in the databases parameter. To communicate with another database the programmer will have to specify the connection change within each script needing it. This can be done with the db command or with the db modifier in any command using SQL.

[0107] There are other variables allowed in this file that will enable verbose debugging features of BrowZ Script. To enable these features you should set the object and either the debug or trace variable to 1. The debug variable is less verbose, but more useful. The trace variable is only useful if you have the BrowZ Script source code. It will show you every function that was called. The recent debug/trace log, if any, will be attached to the bottom of the each script output. This log is also appended to a file called BZScript.log on the server. This log file is cleared each time BrowZ Script is restarted. The example below turns on all debug and trace features in BrowZ Script. It is strongly recommend, however, that you never use all of these together. open.debug and var.debug are usually the most useful.

```
[0108] open.debug=1 # Logs all statement executions
[0109] open.trace=1 # Logs all internal statement
related function calls
```

```
[0110] var.debug=1 # Logs all input and temporary
variable accesses
```

```
[0111] var.trace=1 # Logs all internal var related
functions calls
```

```
[0112] prop.debug=1 # Logs all property variable
accesses
```

```
[0113] prop.trace=1 # Logs all internal property
related function calls
```

```
[0114] session.debug=1 # Logs all session variable
accesses
```

```
[0115] session.trace=1 # Logs all internal session
related function calls
```

[0116] One last thing, the properties file is not limited to just the variables detailed above. You may place many other parameters in this file, and retrieve them from any script in the same directory, using the prop object. See section 3.5.3 for more information on prop object.

[0117] 3.0 Language

[0118] This section details the language of BrowZ Script. To begin with, you need to know that BrowZ Script files are usually just normal HTML files, containing one or more BrowZ Script commands. These commands may be placed anywhere within the HTML document, but it is usually wise to place them in such a way that you do not break any of the HTML syntax rules. This recommendation becomes a requirement if you use an HTML page editor, such as Microsoft FrontPage, Adobe PageMill, or Netscape Composer to add or edit your BrowZ Script files.

[0119] BrowZ Script is designed to be case insensitive, but this has not been thoroughly tested yet. Only fully lower case commands, functions, objects, and variables have been tested.

[0120] 3.1 Syntax

```
[0121] '{'['!'<html tag>]<command>['#'<name>]
[<parameters>]'}
```

[0122] 3.1.1 Brace Delimiters

[0123] Commands are delimited by curly brace characters, '{' and '}', at the beginning and end of each. When you're debugging, always make sure that these braces appear at the beginning and end of each command.

[0124] Note: In BrowZ Script 1.0 the tilde '~' character was used to delimit BrowZ Script commands. These were harder to read and match up when lots of commands were placed close together, so the change was made to curly braces. However, the tilde characters are still supported in 1.2, and you may see some older BrowZ Script files using them. But, these characters will not be supported in future versions of BrowZ Script so it is strongly recommended that you use the preferred curly braces, and begin fixing all 1.0 BrowZ Script files.

[0125] 3.1.2 HTML Tag Stripping

[0126] Following the open brace is an optional exclamation point, '!', and an html tag name. BrowZ Script always strips the entire command from the output. However, in order to maintain the normal rules of HTML sometimes you have to place commands within a paragraph or table cell.

There are HTML tags that get added to the document by the editor which surround the BrowZ Script command. However, you usually do not want these extraneous tags to appear in the output. You can specify that you wish them stripped by adding the exclamation point, '!', and the name of tag you wish to be stripped. Sometimes there is more than one tag that surrounds your command. An example of this is when you place a command in a cell of a table. If you want the entire row removed, rather than just the cell you are in, you would use '!tr' as the tag to strip. All tags and text including the one you specify the BrowZ Script command and those following the command until the trailing tag will be stripped.

[0127] In order for this to work properly you must specify the tag using the same case as the HTML tag. HTML is not case sensitive but BrowZ Script is in this instance. Also, there must be a matching trailing tag. Some page editors do not add the trailing tag in some instances but this is required for BrowZ Script stripping, and it follows that the case of the trailing tag must match also. The best and sometimes only way to use this feature properly is to switch to HTML mode temporarily to see what the first tag you wish to strip is and to note its case.

[0128] Here is an example to illustrate how this stripping works:

[0129] `<p>{!p while select * from MyTable}</p>`

[0130] In this example the matching paragraph tags as well as the BrowZ Script command will be stripped from the output. Since while commands output no text by themselves the output will contain no evidence that the command was even there. Here is another example:

[0131] `<table><tr><td>To Do List</td></tr>`

[0132] `<tr><td><p>{!tr while select * from PROJECTS where PROJ='My Project'}</p></td></tr>`

[0133] `<tr><td>{[row.ACTION]}</td></tr>`

[0134] `<tr><td>{!tr end}</td></tr></table>`

[0135] In this more complicated example the programmer wished to completely remove the rows containing the if and end commands, not just the commands themselves. Also, by using the while command BrowZ Script will repeat everything between the while and end commands for each record found in the database, which in effect adds a row for each. The output of this script would look something like this:

[0136] `<table><tr><td>To Do List</td></tr>`

[0137] `<tr><td>Write the code</td></tr>`

[0138] `<tr><td>Test the code</td></tr>`

[0139] `<tr><td>Write the manual</td></tr>`

[0140] `<tr><td>Prepare the packaging</td></tr>`

[0141] `<tr><td>Ship it!</td></tr>`

[0142] `</table>`

[0143] If the programmer had not put in both '!tr' options the output would have looked like the following, which is definitely not correct HTML:

[0144] `<table><tr><td>To Do List</td></tr>`

[0145] `<tr><td><p><p></td></tr>`

[0146] `<tr><td>Write the code</td></tr>`

[0147] `<tr><td><p></td></tr>`

[0148] `<tr><td>Test the code</td></tr>`

[0149] `<tr><td><p></td></tr>`

[0150] `<tr><td>Write the manual</td></tr>`

[0151] `<tr><td><p></td></tr>`

[0152] `<tr><td>Prepare the packaging</td></tr>`

[0153] `<tr><td><p></td></tr>`

[0154] `<tr><td>Ship it!</td></tr>`

[0155] `<tr><td></td></tr>`

[0156] `</table>`

[0157] Of course you can avoid this entire problem by simply using a text editor rather than an HTML editor, and disregard the rules of HTML. In this case you would have written the following simpler script to produce the same output:

[0158] `<table><tr><td>To Do List</td></tr>`

[0159] `{while select * from PROJECTS where PROJ='My Project'}`

[0160] `<tr><td>{[row.ACTION]}</td></tr>`

[0161] `{end}</table>`

[0162] Don't try to load this script into an editor which knows HTML, because it will definitely complain, or worse, attempt to correct the problem. The command itself is not optional. Also commands are not case sensitive. Each command is itemized in section 3.2.

[0163] 3.1.3 Name

[0164] You may optionally name any statement using a single word or number. This is used when you need to refer to the results of a previous SQL command, after executing another, but is not needed to refer to the results of the current SQL command. See the row object for more information.

[0165] Here's an example:

[0166] `<p>List of items found:</p>`

[0167] `<p>{!p while #tb select * from TABLE}</p>`

[0168] `<p>{!p if select * from ANOTHER where A_STATE='[row.T_STATE]'}</p>`

[0169] `<p>{[tb.T_NAME]}</p>`

[0170] `<p>{!p end}`

[0171] `<p>{!p end}`

[0172] In this example we're printing the T_NAME field from TABLE, but only if the value of that record's T_STATE field is found somewhere in the A_STATE field of the ANOTHER table. If we didn't use the #tb name, we would only have been able to print fields from the ANOTHER table, using the row object. See section 3.5.4 for an explanation of the row object. We could have accomplished the same thing by storing the T_NAME field into a temporary variable, using the var object, so we could access it from within the if command. See section 3.5.6 for an explanation of the var object.

[0173] Note also the use of the row object within the parameter of the if. Access to fields of the row object does not change until the closing brace of the next command containing a statement. This means that row continues to refer to the current record of the while command until the closing brace of the if command, at which point it changes to the row returned from the result of the if. Row again switches back to the row from the while command after the first end, which ends the if and closes the statement's results. See section 3.2.5 for more details on the end command.

[0174] 3.1.4 Parameters

[0175] The parameters to a command vary depending on the command used. Usually, the parameters are a statement, such as SQL. Note, one or more expressions may be used anywhere within the parameters section. See section 3.4 for information on how to add and use expressions of objects and functions.

[0176] 3.1.5 Special Characters and Sequences

[0177] Since HTML empowers certain characters with special meanings BrowZ Script has to be flexible in its interpretation and use of these. If your HTML editor inserts a < instead of a less-than character '<' into your SQL statement, BrowZ Script will properly convert it to a less-than character '<' before attempting to execute it. The following are all of the automatic translations made by BrowZ Script.

[0178] %XX

[0179] A percent sign '%' followed by two hex digits will be converted into the corresponding ASCII character.

[0180] <, >, , &

[0181] These will be converted into less than '<', greater than '>', space '&', and ampersand '&', respectively.

[0182] \{, \}, \}, \}

[0183] When you need BrowZ Script to ignore any special meaning of a character in your scripts simply precede the character by a backslash '\'.

[0184]
, carriage return and line feed

[0185] Carriage returns and/or line feeds within commands will be converted to a space. This is because HTML editors insert spaces in arbitrary positions within long lines. However, if you really do want carriage returns within a command, such as within quoted text, use an HTML line break,
. All
 tags found within commands will get converted to a carriage return and line feed.

[0186] 3.2 Commands

[0187] Here is a list of commands in BrowZ Script, grouped by relationships.

Flow Control	Inter-file	Assignment	File
Do-end	Insert	Set	Copy
If-else-end	Redirect	Unset	
While-else-end	Target	Default	
Db	Source		

[0188] 3.2.1 Do Command

[0189] Syntax: do <statement>, [end]

[0190] The do command executes a statement unconditionally. Any statement may be executed, including SQL, scan, dir, and array statements. See section 3.3 for more information on statements. The do command produces no direct output. A matching end command, which should follow the do command somewhere in the script, closes the statement. It is not necessarily required to have a matching end command because all open statements will be automatically closed when a script completes, but it is better to explicitly end (close) each statement when you are done with it. The end command is required if you need to access results from an outer statement, if or while command when you nest more than one. Here is an example of the use of the do command.

[0191] <p>{!p do insert into MyTable (T_NAME, T_PHONE, T_DATE) values ([quote(var.name)], [quote(var.phone)], [date(sql,today())])}{end}</p>

[0192] <p>A new record was added to the database.</p>

[0193] Note: in BrowZ Script 1.1 and previous this command was named sql. In 1.2 sql is still allowed but is deprecated in favor of do, because statements are no longer limited to just SQL any more. Please use do instead of sql and fix your old scripts, because a future version of BrowZ Script will not support the sql command.

[0194] 3.2.2 If Command

[0195] Syntax: if [<statement>['expression ']], [else], end

[0196] The if command executes a statement unconditionally. It also controls the output and execution of HTML and commands that follow it. If the statement or expression results in a true condition all html and commands after the if until the matching else and/or end command are executed and output. A matching else command is not required, but if used, the HTML and commands between the else and the end are executed and output if the condition is false.

[0197] The condition is based on the results of the statement or the value of the expression. The condition will be true if a statement results in 1 or more rows, and false if it results in no rows. The condition will be true if an SQL insert, delete, or update statement results in 1 or more rows being modified and false if no rows were modified.

[0198] If you specify an expression as the parameter to if then the condition is based on the results of the expression. If the expression results in a number, then 0 or a negative number mean false, and 1 or greater mean true. If the expression results in a string then a null or zero length string mean false, and a string containing 1 or more characters mean true, including the word "false" so be careful. If the expression results in any other object, then a null means false, and any object means true. Here is an example of the use of an if command.

[0199] <p>{!p if update MyTable set T_NAME=[quote(var.name)] where T_ID=[var.id]}</p>

[0200] <p>Record #{{[var.id]}} has been saved.</p>

[0201] <p>{!p else}</p>

[0202] <p>Error: Record #{{[var.id]}} does not exist.</p>

[0203] <p>{!p end}</p>

[0204] If you do not include a parameter of any kind then the condition is based on the previous statement executed, such as a previous do, if, or while command, though the most common of these is the do command. This may be useful when you wish to execute a command previous to where you want to conditionally execute other commands or HTML.

[0205] 3.2.3 While Command

[0206] Syntax: while [<statement>|['expression ']], [else], end

[0207] The while command is similar to the if command, except that it repeats in a loop until the condition becomes false. It also increments the row in the results of the statement when a statement returns more than one row. Everything after the while command to the end or else will be output or executed for each row returned in the result. If there is an else and the statement results in an empty table or no rows affected then everything after the else command to the end command is output once, the same as an if command. If an expression is used as the only parameter then the while will continue as long expression evaluates to true. See the if command for details on how objects are evaluated to a true or false. Here is an example of the use of the while command.

```
<p>{!p while select * from MyTable where T_NAME like
'%'<var.name%>'</p>
<p>Name: {[row.T_NAME]}, Phone: {[row.T_PHONE]}</p>
<p>{!p else}</p>
<p>No names were found containing '{<var.name>}'.</p>
<p>{!p end}</p>
```

[0208] If no statement or expression is included then the while will be executed based on the result of the previous SQL executed in the file. This is useful, for instance, when you want to execute part of your script a single time, if any rows are returned, and part of your script repeatedly, for each row returned. Below is an example of this:

```
<p>{!p if select * from MyTable where T_NAME like '%'<var.name%>'
Here is the select for both the if and while commands.</p>
<table><tr><td>Name</td><td>Phone</td></tr>
<tr><td width=2>{!tr while} Notice the lack of a statement
here.</td></tr>
<tr><td>{[row.T_NAME]}</td><td>{[row.T_PHONE]}</td></tr>
<tr><td width=2>{!tr end} This ends the repeated part.</td></tr>
<tr><td width=2>{[row]} names were found.</td></table>
<p>{!p else} Matches the if.</p>
<p>No names were found containing '{<var.name>}'.</p>
<p>{!p end} This ends the if.</p>
```

[0209] 3.2.4 Else Command

[0210] Syntax: if else end

[0211] Or: while else end

[0212] The else command is used only in conjunction with the if or while commands. It is always optional and specifies where to start outputting if the condition of the if or while is initially false. Any parameters added to this command are ignored so the parameter may be used as a comment.

[0213] 3.2.5 End Command

[0214] Syntax: do end

[0215] Or: if else end

[0216] Or: while else end

[0217] The end command is used only in conjunction with the do, if, or while commands. For all of these commands end specifies the end of the conditional or repeated output. It closes the statement, if any were opened in the matching do, if, or while command. In the case of the while command, it closes the statement once the last row has been retrieved and the last loop has completed. End is recommended, but not necessarily required, for use with the do command, because all statements are closed when the script completes. However, if you nest more than one do, if or while you will be required to have an end statement before you can access results from the outer command. Any parameters added to this command are ignored so the parameter may be used as a comment.

[0218] 3.2.6 Db Command

[0219] Syntax: db <database>

[0220] The db command switches the current database to the one you specify by name. This name must match the name you used in the properties file. All following SQL statements will be executed with the new database. Individual SQL statements may override this by specifying a database explicitly. See section 3.3.4 for more information on specifying a database from within an SQL statement.

[0221] Note: if you used the single database format for the properties file then the db command will not work and should not be used.

[0222] 3.2.7 Insert Command

[0223] Syntax: insert <file path>

[0224] The insert command loads and runs another script. All objects, variables and their values are available to the inserted script. The remainder of the current script is also run after the inserted file executes, except in the case of redirect. See section 3.2.8 for more information on redirect. Any variables that were modified in the inserted script will keep their modified values after the insert completes and the current script continues. Insert is somewhat similar to a function or subroutine call in other languages. Inserted scripts may also insert other scripts, to as many levels as you desire. This command is useful when you need to execute and/or output the same or similar things from more than one script.

[0225] 3.2.8 Redirect Command

[0226] Syntax: redirect <file path>

[0227] The redirect command loads and runs a script from another file and aborts the current script. All objects, variables and their values are available to the redirected script. Any output from the current script already created is discarded and not sent to the client. If the current script had been executed via an insert from yet another script, that insert is aborted also. In other words, the inserting script will

not continue if a redirect occurs in the inserted script. In fact, redirect aborts all multiple levels of insert. There is no problem, however, for a redirected script to redirect or insert other scripts, as you so desire.

[0228] 3.2.9 Target Command

[0229] Syntax: target [`<file path>`]['.`<file extension>`]

[0230] The target command tells the execution engine to create a file and write the output of the current script to it, once the script completes. The output is not sent to client, but once the new script file is written this new script is then executed itself, and any output it generates will be sent to the client.

[0231] It is possible, and sometimes very useful, for the generated file to also, contain BrowZ Script commands. These will execute the same as if the file had been requested directly by the client. One very nice use for the target command is in the caching of dynamic pages, which change much less often than they are requested. In the case of caches, it is also useful to generate a source command to be placed in your target script referring back to the source script. This ensures that the target stays synchronized with the source. See section 3.2.10 for more information on the source command.

[0232] The parameter of this command is either a file path or a dot extension. The file path may be absolute or relative to the currently executing scripts folder. If you use a dot extension parameter then the new script will have the same name as the current one, but with the extension changed to what you specify.

[0233] The target command may be placed anywhere within the currently executing script. If it is placed within a conditional part of your script, such as between if, else, or end then it will only take effect if the condition allows the target command to actually execute. If it is skipped, for whatever reason, the current output will be sent to the requesting client and no file will be written, just as if the target command was not there.

[0234] I have not tested this yet, but it should be possible to create a one-shot script, a script that overwrites itself. When the script executes it changes itself, permanently. A good application of a one-shot script is left as an exercise to the reader.

[0235] 3.2.10 Source Command

[0236] Syntax: source [`<file path>`]['.`<file extension>`]

[0237] The source command tells the execution engine that the current script was created by another script. BrowZ Script will check the dates of the two files. If the current script is older than the source script then the current script is aborted and the source script is executed instead. Its is mainly useful when the script contains a target command. It allows the target script to refer to its source to maintain synchronization between the two files. See section 3.2.9 for more information on the target command.

[0238] The parameter of this command is either a file path or a dot extension. The file path may be absolute or relative to the currently executing scripts folder. If you use a dot

extension parameter then the source script will have the same name as the current one, but with the extension as you specified.

[0239] 3.2.11 Set Command

[0240] The set command will set the value of a variable. Once set, a variable holds and returns its value for the rest of the script, or until the script sets it again. Input and temporary variables are accessed through the var object. Currently, you may only set variables of the session or var objects. If you do not specify which object the member belongs to, the var object is assumed.

[0241] set `<[object.]variable>[=]['<expression>']`

[0242] This variation sets a variable with the value returned by the expression. The brackets are required and signal BrowZ Script to not execute a statement.

[0243] set `<[object.]variable>[=]<statement>`

[0244] This variation sets a variable based on the result of an SQL statement. The variable will be set with the value in the first column of the first row returned. All others columns and rows will be ignored. If an empty result set is return from the statement then the variable will be set to null. The statement is automatically closed so an end command is not needed, or allowed. This variation of set is really just a short cut and could be rewritten as follows:

```
[0245] {unset variable}{if <statement>}{set variable=[row.1]};end
```

[0246] 3.2.12 Default Command

[0247] Syntax: default `<variable>['<expression>']`

[0248] Or: set `<variable><select statement>`

[0249] The default command is almost the same as the set command. It differs only in that it does not change the value of the variable if the variable already has a value assigned. It will only set the value of the variable if the variable is null or is not set.

[0250] 3.2.13 Copy Command

[0251] Syntax: copy `<source file path><target file path>`

[0252] The copy command simply copies one file to another. The file names for the source and target must be specified. The path information may be relative to the current script's folder or absolute. BrowZ Script makes a copy of the file and also sets the modified date to be identical with the source.

[0253] 3.2.14 Expression Command

[0254] Syntax: `['<expression>']`

[0255] The expression command simply outputs the results of the expression. It has no command name, just the expression itself, in square brackets, '[' and ']'. See section 3.4 for more information on expressions.

[0256] 3.3 Statements

[0257] Statements are the workhorses of BrowZ Script; you will use these to do most of your work. Statements are the parameters to the following BrowZ Script commands: do, if, while, set and default. Statements include all of SQL, plus several BrowZ Script only additions. Statements may

be modified in a way unique to BrowZ Script. BrowZ Script expressions may be used anywhere within the statement to make changes, customize, and parameterize it. Plus, there is a db modifier for SQL statements, used to switch connections between more than one database engine.

[0258] Here are the different types of statements:

Statements	Description
Array	Iterates through an array, a variable which holds more one or more values
Dir	Iterates through a directory of files
Scan	Parses a string, iterates through the words
(SQL) select	Iterates through a result set returned from a database
(SQL) insert, delete, update, etc.	Makes changes to a database, or does other database related SQL

[0259] 3.3.1 Array Statement

[0260] Syntax: array <variable>

[0261] The array statement is used to query and iterate (repeated) OR sequence (once) through the values of a variable. All variables in BrowZ Script may contain 0 or more values. If the variable has no value, then the statement returns an empty result set. If the variable contains one or more values then each value will be accessed sequentially through the row object. This statement is usually used only in a while command. However, in an if command, the condition will be true when the variable has one or more values, and false otherwise.

[0262] Here are the fields available through the row object when using this statement with do, if, or while:

Index	Fields	Description
1	Item	The value

[0263] 3.3.2 Dir Statement

[0264] Syntax: dir <folder-path>

[0265] The dir statement is used to enumerate the files within a folder of the file-system. The path should be specified relative to the directory containing the executing script. Or it may be an absolute path, starting with the root directory. The path syntax can be based on the POSIX standard (Unix), using forward slashes, or it can be platform specific, such as with backslashes on Windows.

[0266] Information about each file will be returned as the result-set of the statement. This statement is usually used only in a while command. However, in an if command, the condition will be true when the folder has one or more files, and false otherwise.

[0267] Here are the fields available through the row object when using this statement with do, if, or while:

Index	Fields	Description
1	File	A string containing the full file name
2	Name	A string containing the file name, without the extension
3	Ext	A string containing the file extension, without the period
4	Size	An integer containing the file size in bytes
5	Date	A date containing the last modified date of the file

[0268] 3.3.3 Scan Statement

[0269] Syntax: scan [<delimiters>]<string>

[0270] The scan statement is used to parse a simple, delimited string. A sequential list of words will be returned as the result-set. If no delimiters are specified then all white space is used as the delimiter. Delimiters are specified as one or more non-alpha-numeric characters, followed by a space and the string to parse. The only special case delimiter is the underscore ‘_’. If you put an underscore as the first delimiter then whitespace will be added as part of the delimiters. An underscore may be a delimiter if you put one or more other delimiters first.

[0271] This statement is usually used only with a while command. However, in an if command, the condition will be true when the string has one or more words, and false otherwise.

[0272] Here are the fields available through the row object when using this statement with do, if, or while:

Index	Fields	Description
1	Word	A string containing the word
2	Delim	A string containing the delimiter(s) encountered following the word

[0273] 3.3.4 Select Statement (and all result-set SQL)

[0274] Syntax: [db <database>]select <select clauses>

[0275] This includes all variations of the SQL select statement and any other SQL that returns a result-set. BrowZ Script sends the entire statement (after expression evaluation) to the current database. The current database is the latest database specified with a db command, or the first (or only) database specified in your BZScript.properties file. This statement is usually used with a while command. However, in an if command, the condition will be true when the result-set contains one or more rows, and false otherwise. If this is used with a set or default command, the variable will be assigned the first value returned, the first column and first row. All other columns and rows will be ignored.

[0276] 3.3.5 All other SQL Statements

[0277] Syntax: [db <database>]<sql statements>

[0278] This includes all other SQL statements, including insert, delete, update, those that do not return a result set. The SQL is sent, same as above, to the current database.

These types of statements are usually used with do, if, set, and default commands. Many, but not all, SQL commands will return a value, usually the number of rows modified. This number will be used in the condition of the if command, 0 meaning false, and all other values meaning true. When used with set or default, the value returned will be stored in the variable. It does not usually make sense to use this type of statement with a while command.

[0279] 3.4 Expressions

[0280] Syntax: `['[object [object.] function(<expression>[<expression> . . .]) ']`

[0281] Expressions are designed to return a value. They may be placed anywhere within the parameters of a command, such as within a statement. Or, the expression may be the entire command, (see section 3.2.14). Expressions are delimited by square brackets, '[' and ']'. They can be complex or simple. BrowZ Script 1.2 does not support in-line operator evaluation, such as '+', or '-', though this is planned for a future release. You may use any combination objects, object members, object functions and global functions. But, without in-line operators, you can only pass results from one expression as a parameter to a function, and nesting these function calls as deep as makes sense.

[0282] 3.5 Global Objects

[0283] There are several objects that are accessible globally, to be used in any expression. Objects may have member variables, functions or both. The members of an object can only be retrieved (read), except for the var and prop objects. Use the set or default command to change member variables of prop and var. Here is the list of the global objects:

Object	Purpose
Init	Access initialization parameters of BrowZ Script
Open	Access the results of a named statement
Prop	Access variables stored in the BZScript.properties file
Row	Access the fields, or columns of the current open statement
Session	Access the variables associated with a user's session
Var	Access HTTP POST and URL input, and temporary variables

[0284] 3.5.1 Init Object

[0285] The init object allows you to access the servlet initialization parameters. These are dependent upon how you initialized the BrowZ Script servlet with your web server. Any parameters you initialized your servlet with may be read, but you may not set the value of members of this object.

[0286] Examples:

init.user	Returns the value assigned to user
init.user.id	Returns the value assigned to user.id

[0287] 3.5.2 Open Object

[0288] The open object allows you to access result sets for those SQL statements that you gave names to in your script. If the statement has a result set you may access the columns

by name or number, with 1 referring to the first column. If the statement does not have a result set, you may only retrieve the number of rows affected. This object is not normally needed because you may access statements by name alone. The time this object is needed is if you name a statement same as one of the other global objects. You may not set the value of members of this object, only read them. See section 3.1.1 for information on giving a statement a name.

[0289] Examples:

open.abc.user	Returns the value in column user from statement named abc
Abc.user	Same as above
open.def.2	Returns the value in column 2 from the statement named def
Def.2	Same as above
open.ghi	Returns the number of the current row from the statement named ghi, if the statement returns a result set, i.e. select
open.ghi	Returns the number of rows affected from the statement named ghi, if the statement does not return a result set, i.e. insert, update, or delete
Ghi	Same as the above two examples

[0290] 3.5.3 Prop Object

[0291] The prop object allows you to access variables in the BZScript.properties files. You may not set the value of members of this object, only read the values that you placed in the properties file. See section 2.0 for information on the purpose of this file. Examples:

prop.databases	Returns the value of the databases parameter
prop.abc	Returns the value of the abc parameter
prop.abc.jkl	Returns the value of the abc.jkl parameter

[0292] 3.5.4 Row Object

[0293] The row object allows you to access result sets for the current open SQL statement. If the statement has a result set you may access the columns by name or number, with 1 referring to the first column. If the statement does not have a result set, you may only retrieve the number of rows affected. You may not set the value of members of this object, only read them.

[0294] Examples:

row.user	Returns the value in column user from the current open statement
Row.2	Returns the value in column 2 from the current open statement
Row	Returns the number of the current row from the current open statement, if the statement returns a result set, i.e. select
Row	Returns the number of rows affected from the current open statement, if the statement does not return a result set, i.e. insert, update, or delete

[0295] 3.5.5 Session Object

[0296] The session object allows access to variables associated with a user session. A new session with no member variables is created each time a new user requests a web

page from BrowZ Script. The session stays alive as long as the same user continues to hit the server with the same browser. If the user exits the browser and launches it again a new session will be created, with no initial variables. These variables may be both read and written. This is where you would normally put variables that need to be passed between and among multiple scripts, but not among more than one user. If you need more permanent or persistent variables you should use records in your database.

[0297] Examples:

session.user	Access the variable called user associated with the user's session
session.name	Access the variable called name associated with the user's session

[0298] 3.5.6 Var Object

[0299] The var object is the most used object in BrowZ Script. It allows access to input variables from a form POST or URL parameter. You do not need to distinguish between a form input variables and URL variables, as only one type is allowed at a time. An HTTP POST only returns form input variables and an HTTP GET only returns URL variables, variables passed after the question mark in a URL. These variables may be both read and written, and you may create and use more variables than were initially passed to the script. This is where you would normally put all temporary script variables.

[0300] Examples:

var.user	Access the variable called user
var.name	Access the variable called name

[0301] 3.6 Global Functions

[0302] There are many functions that can be accessed globally, i.e. without using an object to access them. Below is a summary.

Category	Function	Purpose
Boolean	And	Boolean and
	Not	Boolean negation
	Or	Boolean or
Conditional	Eq	Equality comparison
	Ge	Greater than or equal to comparison
	Gt	Greater than comparison
	If	Conditional, switched result
	Le	Less than or equal to comparison
	Lt	Less than comparison
Conversion	Ne	Not equal comparison
	Date	Date conversion
	Int	Integer conversion
Math	Add	Addition
	Div	Division
	Mul	Multiplication
	Mod	Modulus, remainder
	Sub	Subtraction
Query	IsDate	Date checking
	IsNull	Null checking

-continued

Category	Function	Purpose
String	Concat	String concatenation
	Html	HTML conversion
	IndexOf	Finds sub-string within string
	Length	String length
	Limit	String truncation
	LowerCase	Lower case conversion
	Quote	Quotes strings for SQL
	Strip	HTML tag stripping
	UpperCase	Upper case conversion
Time	Url	Conversion for URLs
	Now	Returns the current time
	Today	Returns the current date

[0303] 3.6.1 Add Function

[0304] add(p1 . . .)_int

[0305] The add function returns the sum of all its parameters. Each parameter is converted to an integer before it is added. In the future support for other types will be included, such as float, date, time, etc.

[0306] 3.6.2 And Function

[0307] and(p1 . . .)_boolean

[0308] The and function returns the boolean and of all its parameters. Each parameter converted to a boolean before its used.

[0309] 3.6.3 Concat Function

[0310] concat(p1 . . .)_string

[0311] The concat function returns the string concatenation of the all parameters. Each parameter is converted to a string before it is used.

[0312] 3.6.4 Date Function

[0313] date([type,]p1)_object

[0314] The date function returns either a string or an int, given various types of input data, and possibly a type parameter. If the type parameter is not specified then "short" is assumed.

[0315] Long

[0316] Returns a string formatted as follows: Weekday, Month dd, yyyy.

[0317] Short

[0318] Returns a string formatted as follows: yyyy-mm-dd.

[0319] Sql

[0320] Returns a string for use in an SQL statement formatted as follows: {d 'yyyy-mm-dd'}. This variation is required for all SQL insert or update statements where the field type is a date or date/time.

[0321] 3.6.5 Div Function

[0322] div(p1, p2 . . .)_int

[0323] The div function returns the quotient of p1 divided by p2. If there are more parameters each is divided into the quotient. Each parameter is converted to an integer before it is used.

[0365] The lt function returns true if $p1 < p2$. A good attempt is made to convert the second parameter to the same type as the first before comparison.

[0366] 3.6.20 Mod Function

[0367] `mod(p1, p2 . . .)_int`

[0368] The mod function returns the remainder of $p1$ divided by $p2$. If there are more parameters each is divided into the previous remainder resulting in new remainder. Each parameter is converted to an integer before it is used.

[0369] 3.6.21 Mul Function

[0370] `mul(p1 . . .)_int`

[0371] The mul function returns the product of all the parameters. Each parameter is converted to an integer before it is used.

[0372] 3.6.22 Ne Function

[0373] `ne(p1, p2)_boolean`

[0374] The ne function returns true if the two parameters are not the same. A good attempt is made to convert the second parameter to the same type as the first before comparison.

[0375] 3.6.23 Not Function

[0376] `not(p1)_boolean`

[0377] The not function returns the boolean negation of $p1$. $P1$ is converted to a boolean before its used.

[0378] 3.6.24 Now Function

[0379] `now()_time`

[0380] The now function returns the current time of day.

[0381] 3.6.25 Or Function

[0382] `or(p1 . . .)_boolean`

[0383] The or function returns the boolean or of all its parameters. Each parameter converted to a boolean before it's used.

[0384] 3.6.26 Quote Function

[0385] `quote(p1)_string`

[0386] The quote function returns $p1$ as a string with single quotes around it, or the word NULL if $p1$ is null.

[0387] 3.6.27 Strip Function

[0388] `strip(links, p1)_string`

[0389] The strip function returns $p1$ with all hyperlinks URLs, domain names and image tags removed from it. $P1$ is converted to a string before it is used.

[0390] 3.6.28 Sub Function

[0391] `sub(p1, p2 . . .)_boolean`

[0392] The sub function subtracts all other parameters from $p1$ and returns the result. Each parameter converted to an int before it's used.

[0393] 3.6.29 Today Function

[0394] `today()_date`

[0395] The today function returns the current date.

[0396] 3.6.30 UpperCase Function

[0397] `upperCase(p1)_string`

[0398] The upperCase function converts all characters of $p1$ to upper case and returns it.

[0399] 3.6.31 Url Function

[0400] `url(p1)_string`

[0401] The url function formats $p1$ as a string with characters that are not allowed in a URL converted to % hex. Use this when you need to put a field value into an HTML URL.

[0402] `xor(a, b), neg(a)`

[0403] These functions will take the place of in-line operators, which I'm also planning to add, eventually. They will return int, float, or boolean results based on their input parameters.

[0404] `time(<time format>, <value>)`

[0405] The time function returns the current time as a formatted string, or an integer in a few cases. The variation you choose specifies what the result will look like.

[0406] short: 2:32 PM (This is the default)

[0407] medium: 2:32:15 PM

[0408] long: 2:32:15 PM MDT

[0409] hour: 2 (integer)

[0410] hour24: 14 (integer)

[0411] minute: 32 (integer)

[0412] second: 15 (integer)

[0413] ampm: PM

[0414] sql: {t '14:32:15'}

[0415] `float(data, [digits])_float`

[0416] This function converts the data to a floating-point number. If you specify another parameter then the number will be truncated to that number of digits after the decimal point.

[0417] `quotes(string)_string`

[0418] This is the same a quote but it returns its parameter in double quotes, "".

[0419] Thus, as discussed herein, the embodiments of the present invention embrace the field of dynamic computer document creation. In particular, the present invention relates to systems and methods for providing a scripting language that uses script codes in a document formatted for an independent application. While implementation of the present invention is particularly well suited for creating dynamic HTML documents with database information using scripting commands, the present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes that come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed is:

1. In a system that includes a computer device, a method for facilitating the creation of a web page using a script command and a database, the method comprising the steps for:

creating a text-based document;

adding a server-side script command to the text-based document to create a mixed-format scripted document, wherein the server-side script command provides increased functionality to the web page;

using an interpreter to process the scripting command;
selectively accessing a database when the text-based document is requested by a browser; and

providing the web page on a display device.

2. A method as recited in claim 1, wherein the interpreter is a Java servlet.

3. A method as recited in claim 1, wherein the text-based document is a one of:

(i) an HTML document;

(ii) a spreadsheet; and

(iii) a word-processing document.

4. A method as recited in claim 1, wherein the step for using an interpreter to process the scripting command comprises the step for using the interpreter to scan the text-based document for the server-side script command.

5. A method as recited in claim 4, wherein the step for using an interpreter to process the scripting command further comprises the step for using the interpreter to execute the server-side script command upon encountering the server-side script command.

6. A method as recited in claim 5, wherein the step for using an interpreter to execute the server-side script command initiates the step for selectively accessing a database.

7. A method as recited in claim 6, wherein the step for selectively accessing a database comprises the steps for:

(i) processing information from the database;

(ii) returning a result to the browser; and

(iii) stripping the script command from a resulting document.

8. A method as recited in claim 7, wherein the resulting document is an HTML document without an extraneous tag.

9. A method as recited in claim 8, wherein the browser and the display device are at a client computer.

10. A method as recited in claim 1, wherein the step for adding a server-side script command is performed using one of:

(i) an HTML editor;

(ii) a word processor; and

(iii) a text editor.

11. In a networked system that includes a client and a server, a method for facilitating creation of a web page for display at the client, the method comprising the steps for:

initiating a request from a client computer device to a web server for information corresponding to a web page, wherein the web page is represented by a URL address that is located on the web server;

using the web server to locate a document associated with the URL address;

using the web server to check the file extension of the document to determine whether a script command is located in the document;

if the script command is located in the document, sending the document to an interpreter to execute the script command;

using the interpreter to selectively accessing a database; and

sending results to the client computer device for display of the web page on a display device.

12. The method as recited in claim 11, wherein the step for initiating a request is performed by a web browser of the client computer device.

13. The method as recited in claim 11, wherein the document is a text-based document that includes a server-side script command to create a mixed-format scripted document that provides increased functionality to the web page.

14. A method as recited in claim 13, wherein the text-based document is a one of:

(i) an HTML document;

(ii) a spreadsheet; and

(iii) a word-processing document.

15. The method as recited in claim 11, wherein the step for sending the document to an interpreter to execute the script command comprises the step for loading and parsing the script command.

16. A method as recited in claim 1, wherein the interpreter is a Java servlet.

17. A computer program product for implementing within a computer system a method for facilitating the creation of a web page using a script command and a database, the computer program product comprising:

a computer readable medium for providing computer program code means utilized to implement the method, wherein the computer program code means is comprised of executable code for implementing the steps for:

creating a text-based document;

adding a server-side script command to the text-based document to create a mixed-format scripted document, wherein the server-side script command provides increased functionality to the web page;

using an interpreter to process the scripting command; selectively accessing information from a database; and

providing the web page on a display device.

18. A computer program product as recited in claim 17, wherein the step for using an interpreter to process the scripting command comprises the steps for

using the interpreter to scan the text-based document for the server-side script command; and

using the interpreter to execute the server-side script command upon encountering the server-side script command.

19. A computer program product as recited in claim 18, wherein the step for using an interpreter to execute the server-side script command initiates the step for selectively accessing a database.

20. A computer program product as recited in claim 19, wherein the step for selectively accessing a database comprises the steps for:

processing information from the database;

returning a result to the browser; and

stripping the script command from a resulting document.

* * * * *