

event that a first link is not found the n-depth Compressor looks for a second link by returning to **1002**.

[**0056**] **FIG. 11** illustrates an exemplary method in flow-chart form for the list/cache manager. The cache manager usually waits for a connection (**1110**) and determines the request nature (**1120**) e.g., removing, adding or searching. If the item needs to be removed, the item is first found (**1130**) and then removed (**1140**). If the item needs to be added, it is added (**1150**). If the item needs to be found, the current list is searched (**1160**) and a reply is generated for the search request (**1170**). The list is updated in any of the three requests and a log file is generated (**1180**).

[**0057**] The stealth distribution server has the same functionalities as a distribution server in addition to other functionalities, mentioned herein. The stealth distribution server accepts content delivery requests and creates new requests to retrieve data. Similar to the distribution server, the stealth distribution server supports all the standard proxy server features and the editing and compression techniques discussed in the present invention. The difference between the stealth distribution server and the distribution server include two configuration settings, the option to be setup as a circuit proxy and the option to retrieve everything from a single address. The stealth distribution server uses three networks, one protected inside network, one outside network usually connected to the Internet, and an optional dedicated connection to the OCDS network. The stealth distribution server protects the inside server from the outside network and provides increased bandwidth. The stealth distribution server, while servicing the request for content delivery, protects the web server from outside networks. The stealth distribution server will also compress and cache content to increase the existing bandwidth, which is further enhanced by utilizing the dedicated connection to the OCDS network. Requestors requesting content from a web server will request the content from the stealth distribution server which will provide the content in a compressed form from its cache (database). If the content is not available in the stealth distribution server cache, the content is requested from the web server edited, compressed and cached. The content is then sent to the requestor and is also available in future requests by the same requestor or any other requestor. Thus acting as a protecting firewall to the web server. Both the stealth distribution server and the distribution server, mentioned hereinbefore, may be configured to deliver realtime compressed content with the aid of a robot or over time once the content has been requested.

[**0058**] **FIG. 12** is a diagram showing a general description of the steps used to create a compressed image within the instant content delivery system as well as the attributes of the resulting “.trans” image. “.trans” refers to the proprietary image type developed by Transfinity Corporation. The “.trans” image data is organized as byte streams consisting of data chunks that are read sequentially. Each file begins with header information consisting of a file header **1214** followed by an Image Data Header **1216**, Combination Image Descriptor **1218**, and an n-Bit Compression Header **1220**. The image type may be either lossless or lossy or a combination of both types. This is made possible by the statistically based n-bit compressor. “.trans” supports true color from 2 to 32 bits a pixel as well as transparent key color. Grayscale is supported up to 16 bits per pixel. According to an embodiment of the invention the Scanner or the

Editor first identifies an image type **1202**. For example the image is identified as a BMP, JPEG or GIF **1204**. After the image is identified it is decompressed **1206** and reconstructed back into its aboriginal type **1208**. (Note that BMP is listed in step **1208** only by way of example). Thus an image identified as a GIF is decompressed and reconstructed back into a BMP or DIB image. The image is then compressed **1210** using a proprietary compression system such as the compression scheme in the co-pending U.S. patent application Ser. No. 09/631,368. Step **1212** illustrates the basic layout of a “.trans” image file. A file header **1214** begins each file and contains the following information. The first field, the identifier, is always “.trans”. The second field, `tsize [1]` is the size of the header itself. The third field, `WORD offset [1]`, equals the number of bytes from this position of the beginning of the Image Data **1222**. The fourth field, `numimages[n]` states the number of images in the file. “.trans” supports multiple images of multiple types thus facilitating the aforementioned proprietary n-Bit compression system, which may compress part of an image as “.trans” and compress other parts of an image in a different format (e.g. JPEG) . The fifth field, `numloops[1]`, shows the number of loops an animated image will make. The sixth field shows the type of compression used in the image (e.g. JFIFjpeg/Huffman, BMP/n-bit). The filter type is set to 0 and the last field shows the compression ratio. Table 1 present the aforementioned information regarding fields in the file.

TABLE 1

BYTE identifier[5]	/* Always “TRANS” */
BYTE tsize[1]	/* Size of Transfinity file header */
WORD offset[1]	/* Offset to image data */
BYTE numimages[1]	/* Number of images */
BYTE numloops[1]	/* Number of loops */
BYTE type[1]	/* Compression type */
BYTE filter[1]	/* Filter code */
BYTE compressionratio[n]	/* compressionratio */

[**0059**] The second header is the image header, which contains 10 fields. The first field `BYTE tsize[1]` shows the size of the header, which is always 23 bytes. The second field `DWORD isize[1]` is the size of the compressed data in bytes. The third and fourth fields show the height and width of the image. The high and low transparency range of the image is shown by the fifth and sixth fields respectively `DWORD thrange[1]` and `DWORD ltrange[1]`. The seventh field `WORD pause[1]` determines the amount of time in  $\frac{1}{100}$ ths of a second that the decoder should wait before continuing to process an animation. The eighth field `BYTE packed[1]` Specifies what the decoder is to do after the image is displayed. The ninth field `WORD cid size[1]` is a variable and which is used to store the x and y attributes for animation. The last field is `BYTE reserved[1]` is used for background color information.

[**0060**] Table 2 presents the aforementioned information regarding fields in the image header.

TABLE 2

typedef struct imagedataheader	
{	
BYTE tsize[1]	/* Size of image header */
DWORD isize[1]	/* Size of image data */
WORD width[1]	/* Width of image */